Interprocess Communication Protocols for Computer Networks

Stanford Univ.

Prepared For
Office of Naval Research

December 1975

INTERPROCESS COMMUNICATION PROTOCOLS FOR COMPUTER NETWORKS

by

Carl Allan Sunshine

December 1975

Technical Report #105

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. MDA903-76C-0093 and by the National Science Foundation Graduate Fellowship Program.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY · STANFORD, CALIFORNIA

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: TR105	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Interprocess Communication Protocols for Computer Networks	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Carl Allan Sunshine	8. CONTRACT OR GRANT NUMBER(s): MDA903-76C-0093, ARPA Order No. 2494
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Stanford Electronics Laboratories, Stanford University, Stanford, CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 6T10
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, Information Processing Techniques Office, 1400 Wilson Ave., Arlington, VA 22209	12. REPORT DATE: December 1975 / 13. NO. OF PAGES: 276
14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office): Mr. Philip Surra, Resident Representative, Office of Naval Research, Durand 165, Stanford University	15. SECURITY CLASS. (of this report): UNCLASSIFIED / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this report)

Reproduction in whole or in part is permitted for any purpose of the U. S. Government.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

PRICES SUBJECT TO CHANGE

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computer networks, interprocess communication, communication protocol, performance evaluation, reliability, efficiency, interconnection, internetworking, correctness, initialization, throughput, delay, Gateway, routing, addressing, standards.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report focuses on the design and analysis of interprocess communication protocols for networks of computers. Previous research has emphasized system performance at lower levels, within the communication medium itself. This work examines requirements and performance of protocols for communication between processes in the Host computers attached to the communication system.

Both the reliability and the efficiency of protocols are discussed. Reliability involves overcoming unreliable network transmission facilities to avoid loss, duplication, or out-of-order delivery of data. Reliability performance goals are

DD FORM 1 JAN 73 1473
EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 KEY WORDS (Continued)

20 ABSTRACT (Continued)

defined, and the correctness of different protocol mechanisms in achieving these goals is demonstrated. Consequences of protocol failures (Host crashes) and problems of initializing control mechanisms required for reliable communication are also considered.

Efficiency primarily concerns throughput and delay achievable for communication between remote processes. The performance of successively more powerful protocols including error detection, retransmission, flow control, limited buffering, and sequencing is analyzed. Protocol parameters such as retransmission interval, window size, buffer allocation, packet size, and acknowledgement strategy emerge as important factors in determining efficiency. Several graphs showing quantitative performance results for representative situations are included.

An additional section of the report considers the problems of interconnecting heterogeneous computer networks to allow communication between processes in different networks. Topics discussed include global addressing and routing techniques, level of network interconnection, extent of changes required in individual nets, and functions performed by the interface or Gateway between networks.

INTERPROCESS COMMUNICATION PROTOCOLS

FOR COMPUTER NETWORKS

by

Carl Allan Sunshine

December 1975

Technical Report #105

DIGITAL SYSTEMS LABORATORY
Dept. of Electrical Engineering Dept. of Computer Science
Stanford University
Stanford, California

The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either express or implied, of the Defense Advanced Research Projects Agency or the United States Government

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2494, Contract No. MDA903-76C-0093 and by the National Science Foundation Graduate Fellowship Program.

ABSTRACT

This thesis focuses on the design and analysis of interprocess communication protocols for networks of computers. Previous research has emphasized system performance at lower levels, within the communication medium itself. This work examines requirements and performance of protocols for communication between processes in the Host computers attached to the communication system.

Both the reliability and the efficiency of protocols are discussed. Reliability involves overcoming unreliable network transmission facilities to avoid loss, duplication, or out-of-order delivery of data. Reliability performance goals are defined, and the correctness of different protocol mechanisms in achieving these goals is demonstrated. Consequences of protocol failures (Host crashes) and problems of initializing control mechanisms required for reliable communication are also considered.

Efficiency primarily concerns throughput and delay achievable for communication between remote processes. The performance of successively more powerful protocols including error detection, retransmission, flow control, limited buffering, and sequencing is analyzed. Protocol parameters such as retransmission interval, window size, buffer allocation,

packet size, and acknowledgement strategy emerge as important factors in determining efficiency. Several graphs showing quantitative performance results for representative situations are included.

An additional section of the thesis considers the problems of interconnecting heterogeneous computer networks to allow communication between processes in different networks. Topics discussed include global addressing and routing techniques, level of network interconnection, extent of changes required in individual nets, and functions performed by the interface or Gateway between networks.

ACKNOWLEDGEMENTS

First and foremost, I wish to thank my adviser, Professor Vint Cerf, for his constant support, good advice, and good humor throughout the preparation of this work. Few advisers have done their job so well, and allowed graduate study to be so pleasant and profitable an experience. The other members of my committee, Professors Forest Baskett and Gene Golub, also provided valuable input to this work. Their personal concern and counsel over the past three years have been most appreciated. Thanks also go to Bob Metcalfe and Jon Postel for many helpful suggestions on the text of this report, and to Tom Bredt for his support during my first years at Stanford.

My fellow students in the Digital Systems Lab have contributed in many ways to this work, especially Ron Crane, Richard Karp, Jim Mathis, John Mortenson, and Alan Smith. The ready ear and thoughtful comments of my office partner, Yogen Dalal, have been most helpful and encouraging, while his hard work has gotten us over many obstacles. Special thanks also go to Carolyn Taynai, that most human of secretaries, for her exceptional attention to our well-being in all matters.

This research has been financially supported by the National Science Foundation Graduate Fellowship Program, and by the Advanced Research Projects Agency of the Department of

Defense. Facilities and individuals at several ARPA sponsored institutions including the Digital Systems Lab, the Artificial Intelligence Lab, and the SUMEX project at Stanford University, the Augmentation Research Center at Stanford Research Institute, the Information Sciences Institute of USC, Bolt Beranek and Newman, The Department of Computer Science of University College London, and the Rand Corporation have been instrumental in supporting this research and producing this report. Special thanks go to Ray Tomlinson (BBN) for his tireless efforts on the BCPL compiler, to Ed Tompkins (RAND) for polishing the illustrations, and to Ray Finkel (SU-AI) for developing the type font for this report.

IFIP Working Group 6.1 (Internetwork Working Group of the International Federation for Information Processing) has provided an invaluable forum for the exchange of ideas on communication protocols. Louis Pouzin, Roger Scantlebury, Hubert Zimmerman, and Alex McKenzie have participated in numerous stimulating discussions on the topics of this research.

My final thanks go to family and friends who have given me the encouragement to continue my work, and most specially to Tove who has made the best of many long nights at the office and sent me off ready to do battle again the next morning.

TABLE OF CONTENTS

Table of Contents

Table of Contents

Table of Contents

LIST OF TABLES

LIST OF FIGURES

List of Figures

List of Figures

List of Figures

Chapter I

INTRODUCTION

This thesis focuses on the design and analysis of interprocess communication protocols for use in computer networks. The feasability and utility of computer networks has been clearly demonstrated recently with several sophisticated nets fully operational (NPL, ARPA, TYMNET, ALOHA) and many others planned (CYCLADES, EPSS, SITA, CANUNET, AUTODIN II) (see Appendix B). However, much of the research accompanying these developments has emphasized system performance within the communication network itself. Our study examines the requirements for communication between processes in the Host computers attached to the network.

To clarify our level of interest, we note the parallel history of computer network development and single computer system development. In single computer systems, the original empahasis was on "hardware" questions of memories, bus structure, basic arithmetic and logic operations, etc. Eventually such hardware design problems became a specialty, and efforts to provide a more convenient interface to the computer user grew in importance. Programming languages, operating systems, and time sharing were born.

Similarly, the first years of computer network development have emphasized internal design questions such as circuit topology and capacity [Frank70, Frank72a, Cerf75a], routing [Frank71, Fultz72, McQuillan74], switching node requirements [Fultz72, McQuillan72], reliability [VanSlyke72], and congestion control [Davies72, Kahn72]. This emphasis is most apparent in the ARPANET and related packet switching network experience [Frank72, Metcalfe73]. Many of these internal design problems now have a well developed theory and practice [Kleinrock70, Frank72, Pyke73, Karp73, Kershenbaum74] that serves to provide the basic communication facility or transmission medium that interconnects network users at the lowest level.

Unfortunately, processes attempting to communicate with each other over a computer network face a problem similar to humans trying to use a single computer: the basic or "raw" facility provided is often too primitive, unreliable, or otherwise inconvenient. The traditional approach in the computing domain has been to create an operating system to bridge the gap between raw machine and user desires, creating a "virtual" machine that is much more powerful, reliable, and convenient.

To facilitate interprocess communication over a computer network, a similar "augmented" communication facility must be

built upon the basic network services available. In fact, many different levels of augmented service prove desirable for various special communication purposes [Crocker72]. A partially reliable "best effort" communication service represents the lowest level, followed by a general purpose fully reliable interprocess communication protocol, and finally various special purpose services such as file transfer, remote job entry, interactive terminal, and graphics.

Compared to the rigor of internal network design theory and practice, the science of higher level protocol design is in its infancy. This thesis focuses on the general interprocess communication protocols which provide the basic facility on which more specialized services will be built [Pouzin74c].

Although good interprocess communication facilities are a necessary condition for the flexible resource sharing envisioned by network architects [Roberts72, Kahn72a, Watson73, McKay73], they are by no means sufficient. A wide range of higher level problems in distributed system design such as synchronization, file systems [Thomas73], task partitioning, resource allocation, priority assignment [Bowdon72], etc. remain to be solved.

Resource sharing in the distributed environment of computer networks imposes special demands on interprocess communication facilities. Processes are seen as the active

elements in a distributed computing system. Human users at terminals, I/O devices, file systems, service routines, operating systems, are all represented by processes that communicate to accomplish their goals. Processes are often treated as equals for communication purposes rather than requiring a "master" and "slave" relationship typical of polling or centralized control systems (e.g. IBM's SDLC [Donan74, Kersey74]). As opposed to traditional centralized systems where reliability is often taken for granted, the distributed environment of computer networks demands that the interprocess communication facility pay explicit attention to assuring reliability [Metcalfe72]. These considerations determine the type of augmented service desirable, or performance goals for an interprocess communication protocol in a computer network environment.

Reliability and efficiency may be distinguished as two main classes of performance goal. Reliability of the interprocess communication protocol involves avoiding loss or duplication of data transmitted, delivering data in the same order as submitted, and properly initializing and terminating data transfers for continued reliable operation. Efficiency primarily concerns throughput and delay achievable for communication between remote processes. These depend on the operation of protocol mechanisms such as retransmission, flow control, buffer allocation, sequencing, and fragmentation.

These performance goals define one side of the protocol design problem, while the transmission medium characteristics define the other. To the greatest extent possible, we take transmission medium behavior, particularly the difficult characteristics of packet switching nets (PSN), as a given set of characteristics which must be dealt with by a protocol, rather than assuming them away to simplify analysis. After a summary of the contents of this thesis in section 1, we return to discuss transmission medium behavior in section 2. This characterization serves as a basis for protocol design considerations throughout the rest of this work.

1. SUMMARY

Chapters II and III present the major new results of this study concerning reliability and efficiency of protocols respectively. Our contribution is both methodological in developing new analysis techniques, particularly in chapter II, and substantive in presenting answers to protocol design problems. Chapter IV presents a survey of recent work on protocols suitable for the interconnection of packet switching networks and a discussion of the interface or Gateway between networks.

Chapter II defines reliability performance measures and considers the protocol mechanisms necessary to achieve various levels of reliability. Interest in protocol verification has increased recently, with several authors applying various proof techniques to verifying certain aspects of protocol reliability [Postel74, Bochman75, Merlin75]. We employ less formal techniques in order to achieve results for more realistic assumptions about underlying transmission medium behavior. We are particularly interested in developing protocols able to overcome the potentially hostile transmission characteristics of packet switching networks, and our analysis covers the full range of network behavior.

The consequences of protocol failures (for example due to Host crashes) are also considered, leading to an important

result that interprocess communication protocols cannot guarantee "invisible" recovery from protocol failures. That is, loss or duplication of data cannot be prevented by the protocol itself after a failure, but higher level error recovery procedures must be invoked.

Chapter II also considers the problems involved in initializing the control information required by reliable communication mechanisms. Sophisticated methods of selecting initial control information values and synchronizing the other side of a connection prove necessary in a potentially hostile environment such as a PSN. Such mechanisms are defined and verified using a state diagram model which includes the state of both protocol processes (on each side of the connection) plus information transmitted between processes.

Chapter III defines efficiency performance measures of throughput and delay for interprocess communication protocols. Successively more powerful protocols including retransmission, flow, control, limited buffering, and sequencing are analyzed to determine the impact of protocol parameters such as retransmission interval, window size, buffer allocation, and fragment size on efficiency. Transmission medium characteristics such as delay, bandwidth, and errors also strongly affect efficiency. In Chapter III we are forced to make some simplifying assumptions about transmission medium characteristics.

Chapter IV considers the problem of interconnecting independent computer networks to provide communication between processes on computers in different networks. We compare several approaches to network interconnection including those requiring substantial changes to existing local net operations. Techniques to implement internet addressing, routing, and other protocol services discussed in chapters II and III on top of local network facilities appear to be feasible without imposing changes on individual nets.

2. TRANSMISSION MEDIUM CHARACTERISTICS

This section identifies the important characteristics of the basic transmission medium or communication facility that the protocol must use in providing an augmented service. The six characteristics discussed emerged primarily through experience with packet switching neiworks as the basic transmission medium, and are most appropriate to that context. The transmission medium is assumed to accept packets or blocks of data from a source and make a "best effort" to deliver them to the destination. That is, we limit our concern to packet communication media [Metcalfe73], although not strictly to packet switching nets. Many of the points raised apply to other network technologies or even simple communication lines as well, and an effort is made to include these points in the following.

The transmission medium characteristics discussed are:

1) Variable delay
2) Duplication of packets
3) Loss and damage of packets
4) Out-of-order delivery
5) Packet size
6) Bandwidth

Delay

Transmission medium delay is the amount of time between submission of a packet at the source and delivery of the packet at the destination. Traditionally, total delay is decomposed into transmission delay, or the time required to transmit all the bits of a packet at the nominal rate of the transmission medium, plus propagation delay, or the time it takes a bit to travel through the transmission medium to the destination. On a dedicated hardware line these delays are relatively constant. If the transmission medium is shared among many users as is usually the case, there may also be access or queuing delay while a packet waits its turn to be transmitted.

In store-and-forward packet switching networks with many nodes, the packet experiences combinations of these delays at every hop, and as competing traffic level increases, larger access or queuing delays occur at each node. Errors followed by retransmissions between nodes, and alternate routing of packets also contribute to variations in the delay experienced by different packets. Without specifying the global traffic pattern and routing algorithm, it is difficult to detail the delay distribution, but in general variations equal to or greater than the mean delay are typical [Forgie75].

Satellite links with high bandwidths impose large propagation delays (about 250 msec) so retransmission delay

becomes a more important source of variation, although satellite error rates are lower than ground lines [Sastry74]. Loop nets provide relatively constant delay characteristics with access delay accounting for the major variability.

An important delay characteristic for protocol design is the maximum propagation time or packet lifetime in the transmission medium, represented by L. For simple data links and loop nets, packet lifetime is nearly constant and determined largely by line length and physical transmission properties. In packet switching networks with occasional routing anomalies and other malfunctions, L may be orders of magnitude greater than the normal or mean propagation times. As discussed in chapter II, protocols must always be wary of "old" packets arriving, so a minimum L is desirable. One means suggested to achieve this is a packet which self-destructs after a specified time in the net. Further consideration of such transmission medium problems is beyond the scope of this work, and L is taken as one of the given characteristics of the transmission medium.

Duplication

A single packet submitted for transmission may be duplicated by the transmission medium and more than one copy delivered to the destination. Normally the transmission medium removes any duplicates it generates (by internal retransmissions), but certain line or node failures at critical moments can result in duplicates emerging at the destination.

Loss and Damage

A great deal of work has been done to characterize the errors occurring on real transmission lines of various types [Townsend64, Benice64a, Trafton71, Burton72]. Detecting damaged packets is a well developed problem in coding theory. Without further discussion, we assume that the protocol designer has techniques available to decode packets and detect errors in packets to any desired degree of reliability. Davies and Barber (1973) survey the problem of error characteristics and suitable coding techniques. Higher reliability requires longer codes, increasing the overhead to transmit a packet, and reducing effective bandwidth as discussed in chapter III.

On hardware lines, unless the line is completely open, it is safe to assume that a transmitted packet will arrive either intact or damaged. In networks with multiple lines and nodes in the transmission path, the possibility of total packet loss is finite and must be explicitly recognized. Some networks even discard packets as a means of internal congestion control.

Normally a protocol will discard damaged packets and wait for or request their retransmission. This converts the problem of packet damage to packet loss, treating both with the same mechanism (retransmission). However, some applications may tolerate the delivery of damaged packets to the protocol user, and/or loss of some packets.

Ordering

Packets may arrive at the destination in a different order than they were submitted to the transmission medium. In a packet switching net with multiple paths from source to destination and alternate routing of packets, a packet submitted later may travel by a shorter route and arrive sooner than a packet submitted earlier. In line switced nets or on simple data links where all packets follow the same route, retransmission to correct internal errors may cause the retransmitted packet to arrive after one or more later packets transmitted successfully the first time. Some networks provide an ordering facility at the destination, implementing an end-to-end sequencing service within the network communication facility.

Packet Size

The transmission medium normally has a maximum size packet that it will accept. If a process wishes to send a chunk of data larger than this size, the protocol must fragment the chunk into pieces small enough for transmission, and reassemble the chunks, or at least deliver the fragments in order at the destination. The transmission medium itself may have to fragment packets for transmission on certain links (particularly likely if different nets are connected together), so there may be layers of fragmentation and reassembly, or a uniform

fragmentation scheme used at all levels which requires reconstruction only at the destination.

The optimization of packet size involves consideration of line rates, error characteristics, traffic patterns, and performance objectives [Metcalfe73, Crowther74]. At the interprocess communication protocol design level, we assume a given maximum packet size, P.

Bandwidth

The transmission medium accepts packets at a nominal bit rate B. In a packet switching net this rate is usually the hardware line capacity from the Host to the packet switch. However, the transmission medium may become "unavailable" at times due to internal congestion control mechanisms, and the effective bit rate offered is further reduced by framing and control information required on the line. These considerations affect any protocol using the transmission medium, and will not be considered further.

Chapter II

PROTOCOL RELIABILITY

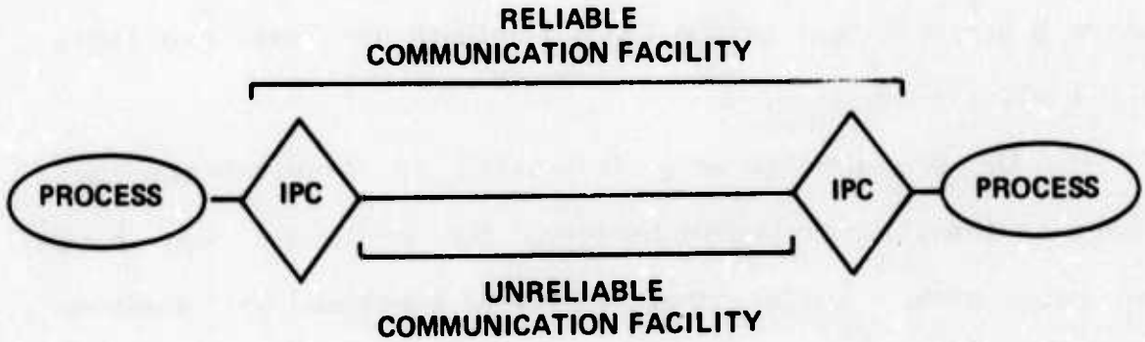
1. INTRODUCTION

In this chapter we study the reliability of interprocess communication protocols. As discussed in chapter I, this requires a clear understanding of protocol performance goals and transmission medium characteristics relevant to reliability since a protocol must bridge the gap between services available and facilities desired.

We are particularly interested in treating the full range of transmission medium behaviour that protocols may have to cope with. While other authors have assumed well-behaved transmission media in order to apply formal analysis techniques, we are more interested in developing protocol mechanisms suitable for worst case situations in real packet switching network environments. We return to this difference in goals in section 1.2 below.

Transmission medium characteristics most relevant to reliability include delay, loss, damage, duplication, and out-of-order delivery of information (cf chapter I). The basic function of an interprocess communication protocol is to mask these undesirable characteristics and provide a reliable and convenient communication path between processes (see figure 1).

FIGURE 1 AUGMENTED SERVICE FROM INTERPROCESS COMMUNICATION PROTOCOL (IPC)



Hence an important goal of protocol analysis is to demonstrate that a candidate protocol or class of protocols indeed provides the desired reliability: i.e. does not duplicate packets, lose packets, or deliver them out of order.

Protocol verification is one part of the complete protocol design process. Successful protocol design also requires a clear specification of performance goals, development of mechanisms to achieve those goals, and evaluation of alternative mechanisms. Evaluation includes verification that the performance goals are met when the protocol is functioning normally. Unfortunately, normal computer system operations are occasionally disrupted by catastrophic failures (hardware faults, deadlocks, protection violations, restarts, etc.). Hence another important consideration of protocol analysis concerns the results of protocol failures. By protocol failures we mean the (rare) malfunction of a normally correct protocol due to some external catastrophe, rather than a protocol that normally functions incorrectly due to a flaw in its algorithms.

A third component of protocol evaluation concerns initialization of the protocol mechanisms used to overcome transmission medium deficiencies. Since this initialization may have to be performed over the same unreliable transmission medium, it presents a difficult synchronization problem.

These four topics are the main subject matter of chapter II:

(1) Definition of performance goals and protocol mechanisms to achieve reliability.

(2) Verification of correct protocol operation under normal circumstances.

(3) Results of protocol failures.

(4) Protocol initialization requirements and techniques.

Whenever possible, the cost of the various solutions to these problems is discussed.

Section 2 outlines the important parts of an interprocess communication protocol, and defines reliability performance measures (efficiency measures are considered in chapter III). Section 3 describes a simple protocol to avoid loss and duplication of packets, verifies the reliability of this protocol, and explores the consequences of protocol failures. Section 4 extends the analysis to a more powerful protocol including a sequencing mechanism that correctly orders delivered packets.

Analysis of the protocols specified in sections 3 and 4 shows that when both sides of the protocol function correctly, loss, duplication, and ordering problems are eliminated. However, when one side of the protocol fails (memory loss) as would occur in a Host crash/restart, we prove that loss or duplication of packets may occur. That is, it is impossible to guarantee error-free recovery after a failure by either side of a connection.

In section 5 we consider the additional problems of initializing the mechanisms which achieve reliable communication. This initialization is widely referred to as connection establishment, and presents a difficult synchronization problem since it must be accomplished using the unreliable basic communication facility. The concepts of connection and connection state are defined, and mechanisms to reliably establish connections (initialize protocols) are specified and classified. We demonstrate the limitations of various mechanisms and prove the robustness of the "3-way handshake" mechanism for connection establishment [Tomlinson74, Dalal74] using a composite state diagram model. Consequences of failures (memory loss) and failure recovery techniques are also considered.

1.1 Related Work

"Closed loop" or "feedback correction" type protocols suitable for overcoming the transmission medium characteristics discussed in chapter I have been widely treated in the literature [Benice64a, Lynch68, Stutzman72, Burton72, Metcalfe73]. The ARQ type protocol is more suitable for hardware lines where compiete packet loss is impossible, since it requires either a positive acknowledgement (ACK) or negative acknowledgement (NACK or Retransmission Request) for every packet sent.

Several authors have used state diagrams to model simple ARQ protocols [Lynch68, Bartlett69, Birke71, Bochman74]. Lynch presents an informal proof that these protocols provide reliable communication over well behaved transmission media that never lose packets, duplicate packets, or deliver packets out of order. Recently Bochman has analyzed the same protocol by the method of action sequences. Seidler (1975) presents a more formal model for analysis of ARQ type protocols.

A Positive Acknowledgement, Retransmission on timeout (PAR) type protocol is more suitable in a packet switching net environment where data packets or acknowledgements may be completely lost, since these protocols do not require a NACK to stimulate retransmission. Forward error correction may be used in addition to error detection on noisy channels to reduce retransmission [Benice64b, Sastry74]. Metcalfe provides an excellent summary of the motivation for and suitability of PAR protocols [Metcalfe73 pp. 3-4 to 3-11]. Kalin (1971) has also discussed several important considerations in protocol reliability including protocol initialization.

Postel (1974) has analyzed some simple examples of PAR protocols for "proper termination." This analysis shows that the specified protocol functions correctly in avoiding loss, duplication, and out-of-order delivery. Postel's work does not treat the general class of PAR protocols or examine the consequences of protocol failures. A specific connection

establishment procedure (the ARPANET Initial Connection Protocol) is shown to have a race condition, but the general question of connection establishment is not treated.

Merlin (1974, 1975) has used Petri nets and their corresponding "token machines" to show that a simple class of PAR protocols is "recoverable" from loss or duplication of packets by the transmission medium, and that packets are delivered in order. Connection establishment and protocol failures are not discussed, although the analysis technique may be applicable to some types of failures.

Bochman (1974) has analyzed some simple PAR protocols using the "action sequences" associated with a state diagram model. He has also explored an algorithmic protocol specification as a basis for both assertion proof techniques and protocol implementation by structured programming [Bochman75].

Le Moli (1973) has proposed a "colloquy" model for protocol specification consisting of a finite state machine with clearly defined user interface on one side and network communication interface on the other side. Danthine and Bremer (1975a, 1975b) have extended this model to facilitate simulation of protocols.

Gilbert and Chandler (1972) have treated the interaction of parallel processes by defining a "composite state" including the state of each process and the values of shared variables. Bredt (1973) has extended this model to allow infinite numbers

of processes or infinite values for variables. The requirement for shared variables between processes prevents the direct application of these techniques to communication protocols. However, in section 5 we have modified this model to allow message-based interprocess communication and have applied the extended model to verify a complex protocol initialization mechanism.

Day (1975) has begun research to determine the issues involved in designing "resilient" protocols. He suggests that protocol specification techniques are of primary importance, and that several approaches to verification may prove useful including formal modeling, program proving, implementation aids, and implementation testers or exercisers.

Members of IFIP WG6.1 (INWG) have been active in developing interprocess communication protocols. Researchers at Stanford University and Bolt Beranek and Newman have been particularly interested in protocol reliability [Cerf74b, Dalal74, Sunshine74, Bennes74a, Tomlinson74, McKenzie74].

1.2 Protocol Specification and Verification Techniques

Analysis and design of communication protocols requires a clear protocol specification. A good protocol specification must ultimately serve several purposes, including definition, verification, simulation, implementation, and documentation of

the algorithms involved [Danthine75b, Bochman75]. We do not attempt to develop a complete theory of protocol specification, or of protocol verification, although both specification and verification are important in the broader performance analysis we seek.

We have used different protocol specification techniques for the different performance topics treated in this thesis in order to most clearly define the protocol behaviour relevant to each topic. Sections 3 and 4 employ a flowchart or algorithmic protocol specification. Section 5 uses a state diagram specification consistent with the exchanges of control information required to initialize and terminate connections. Appendix A develops a detailed protocol specification model based on state diagrams with additional "context" information.

Authors primarily interested in verification have employed formal models such as Petri nets [Merlin74], UCLA Graphs [Postel73], and state diagrams [Bochman74, Lynch68]. By specifying a protocol in terms of one of these formal models, the powerful general theory developed for these abstract models may be brought to bear on a particular aspect of protocol verification. These techniques have succeeded in verifying some facets of protocol reliability assuming reasonably well-behaved transmission media.

Unfortunately, both the complexity of more powerful protocols, and more hostile transmission media are beyond the

capabilities of current formal models. The explosion of states or nodes required to represent more complex protocols causes some problems. Other difficulties arise in trying to incorporate transmission media allowing total loss (as opposed to damage) of packets, large amounts of internal storage, internal duplication, and out-of-order delivery of packets.

We have been forced to abandon some of the rigor of formal definitions and models in order to achieve results of broader scope. Our protocol specification techniques include prose, algorithms, flow charts, and state diagrams. Our proof techniques include decomposition into simple modules, exhaustive or complete test input sets, assertion proving [Floyd67, Naur66, Hoare69], and the formal models mentioned above. An advantage of the informal assertion techniques used throughout this chapter is that in many cases they can be used to demonstrate protocol failure consequences and initialization requirements in addition to correctness under normal operation.

2. PROTOCOL MECHANISMS AND PERFORMANCE GOALS

In this section we present an outline of the mechanisms employed in typical positive acknowledgement, retransmission (PAR) type protocols. The description is purposely broad in order to encompass a wide class of protocols. Sections 3 and 4 examine more detailed examples of PAR protocols.

After outlining protocol mechanisms, we define four performance measures used in later sections to evaluate protocol reliability. These measures relate to loss, duplication, and out-of-order delivery of packets.

2.1 Protocol Definition

A PAR interprocess communication protocol consists of a sending discipline, a receiving discipline, and a transmission medium for sending packets (messages, letters, finite length bit strings) between processes.

The sending discipline accepts packets from a process, attaches any control information used by the protocol to achieve reliable communication, and passes the packet to the transmission medium. (Submitted packets may be fragmented into smaller pieces before transmission.)

Normally the sending discipline will retransmit each packet at intervals determined by a retransmission timeout

parameter R, until a positive acknowledgement (ACK) is received. Then the process is notified that the packet has been successfully delivered. Another parameter, the quit time Q, determines when the sending discipline should give up and report possible failure to deliver a packet.

The receiving discipline receives packets from the transmission medium and uses the control information to eliminate duplicates, reassemble or reorder fragments, and deliver packets to the process in order. Successfully received packets are acknowledged. The transmission medium accepts packets from the sending discipline and delivers them to the receiving discipline subject to the delay, loss, duplication, ordering, size, and bandwidth characteristics discussed in chapter I.

Since a PAR protocol provides bi-directional communication between two processes, a sending and receiving discipline are required on each side of the communication path. The protocol at each side must be initialized (control information set up) as discussed in section 5 before reliable communication can begin.

2.2 Performance Measures

(1) DELIVERY: A protocol *successfully delivers* packets if every packet submitted by the source process is eventually delivered (undamaged) to the destination process. A protocol *fails to deliver* a packet if a packet submitted to the protocol is not delivered (undamaged) to the destination process.

(2) LOSS: A protocol *loses* a packet if it reports successful delivery of a packet to the sending process when in fact the packet has not been successfully delivered to the destination process. A protocol does not lose packets if it reports successful delivery only if the packet was in fact successfully delivered.

(3) DUPLICATION: A protocol *duplicates* packets if a single packet submitted by the source process is delivered more than once to the destination process. A protocol does not duplicate packets if every packet submitted is delivered at most once. (If the process submits the same message twice, both copies will be delivered at the other end--this is not duplication.)

(4) ORDERING: A protocol delivers packets *in order* if packets are delivered to the destination process in exactly the same order that they were submitted by the source process. A protocol delivers packets *out of order* if packets are delivered in a different order than they were submitted.

3. BASIC PAR PROTOCOL

In this section we consider a class of simple PAR protocols without sequencing, fragmentation, flow control, or connection establishment. In particular we note that this class of protocols provides no mechanism for sequencing packets that may arrive out of order.

First we define this class of protocols using the general outline of a sending discipline and a receiving discipline presented in section 2:

SENDING DISCIPLINE (see figure 2):

> Each packet submitted by the source process is assigned a unique identifier. (We temporarily ignore the problems of an infinite ID space.) The packet is transmitted, and a copy is retained.
>
> Arriving ACK's are checked for errors, and damaged ones discarded.
>
> When an ACK referencing this identifier is received, the retained copy is discarded (and the source process notified of success). If no ACK is received within the retransmission timeout period R, the copy is again transmitted and the cycle repeated. If the quit time has been exceeded, retransmission is suspended (and the sending process notified).
>
> ACK's for discarded packets are ignored.

RECEIVING DISCIPLINE (see figure 3):

> Each packet received from the transmission medium is checked for errors and discarded if damaged.
>
> If not damaged, the packet's ID is added to the list of received-packet ID's and an ACK referencing the identifier

FIGURE 2 PAR PROTOCOL SENDING DISCIPLINE

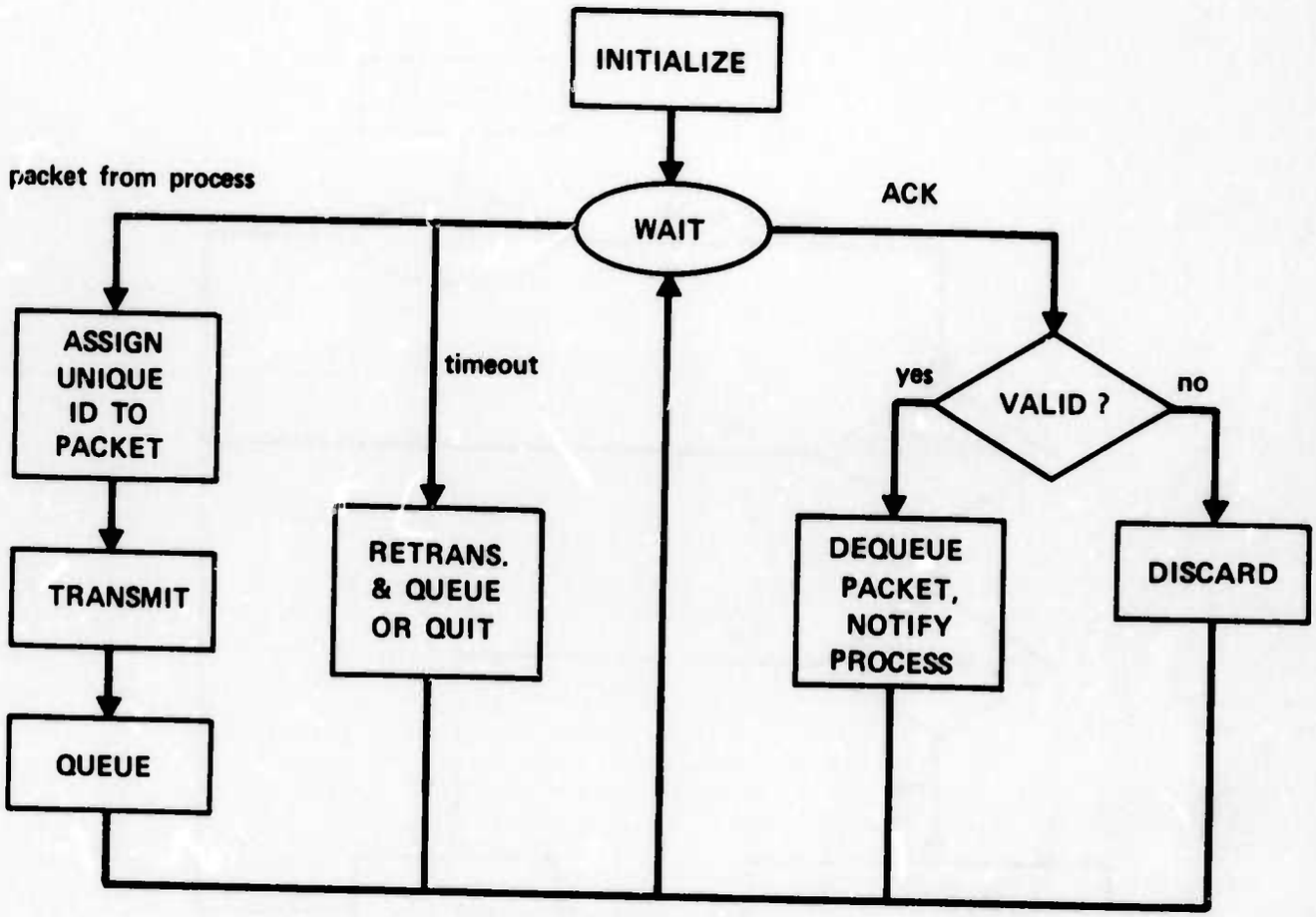
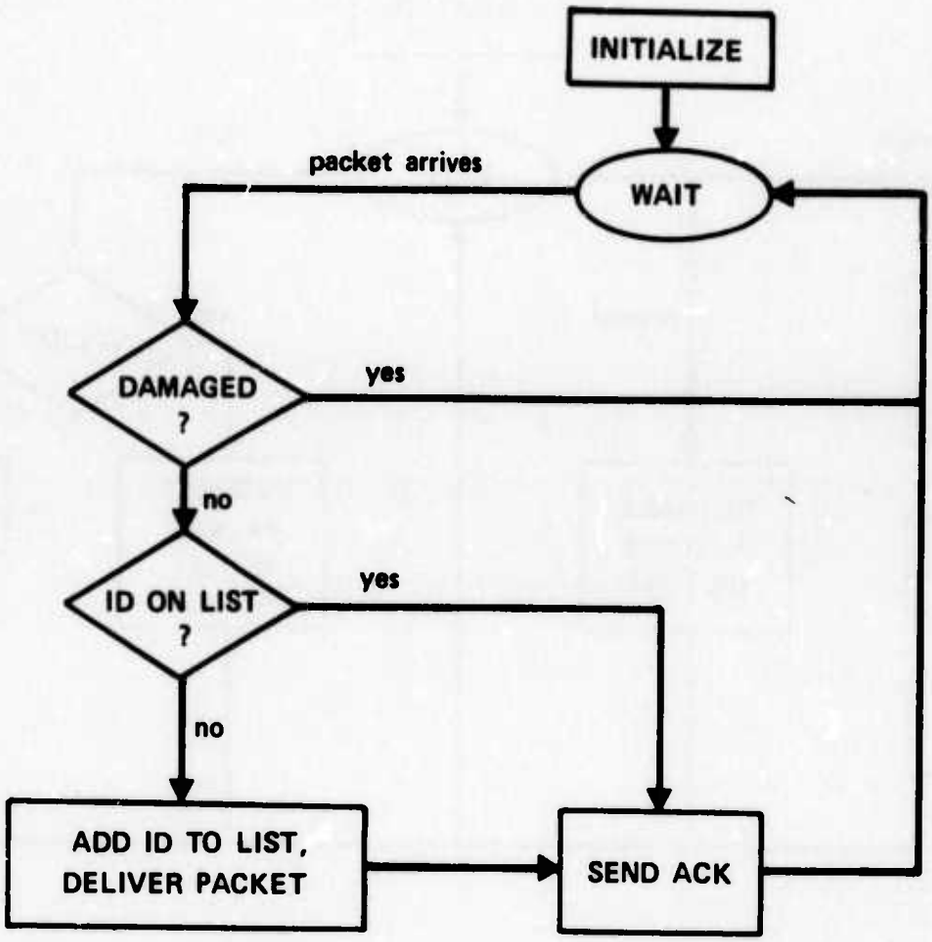


FIGURE 3 PAR PROTOCOL RECEIVING DISCIPLINE



Is transmitted. (We temporarily ignore the fact that the received-packet list size increases as more packets are received.) If the packet's identifier is already in the list, the packet is discarded as a duplicate. Otherwise the packet is delivered to the process.

TRANSMISSION MEDIUM:

Characterized by such parameters as delay, maximum packet lifetime in medium, bandwidth, (non unity) loss probability, and (non unity) damage probability. The complications of addressing, routing, and multiplexing many connections over a single path are ignored here--the protocol is defined for a single connection.

The protocol is initialized when both sides have empty received-packet lists and no packets have been sent. (How to reliably accomplish this is discussed in section 5.)

The above protocol definition assumes that all damaged packets and acknowledgements will be detected. In fact it is not possible to detect all transmission errors, resulting in occasional acceptance of a faulty packet or ACK. However, the probability of an undetected error can be made extremely small at modest cost by use of well known coding techniques, and we will continue to assume perfect error detection.

Although the above protocol definition is quite specific, it still serves to define a class of protocols equivalent for purposes of reliability analysis. Different mechanisms for unique identifier selection, for example, or even additional protocol mechanisms such as negative acknowledgements to stimulate retransmission are included in this class of

protocols. These differences may have important effects on efficiency or cost of implementation, but do not alter the reliability of the protocol.

Having specified a class of simple PAR protocols, we now show that this class satisfies several of the reliability performance goals defined in section 2.

THM 1: A correctly functioning PAR protocol with infinite quit time never loses, duplicates, or fails to deliver packets.

THM 1A: A correctly functioning PAR protocol with finite quit time never loses or duplicates packets, and the probability of failing to deliver a packet can be made arbitrarily small by the sender.

PROOF:

DUPLICATION:

No duplicate packet generated by the sending discipline or transmission medium will ever be delivered to the process, because in checking the list of received-packet ID's, the receiving discipline will discard them.

LOSS AND FAILURE TO DELIVER:

There is a nonzero probability that the transmission medium will successfully transmit a packet. Hence an infinite quit time implies eventual successful delivery with

probability one. (However this may take a long time if the transmission medium is highly unreliable!)

For finite quit times, the time may be exceeded before successful transmission. However, the process is notified that the packet may not have been delivered (it also may have been delivered if the ACK's are lost), and can command the protocol to reset the quit time and continue, or give up. The protocol never reports successful delivery falsely, and the process can make the probability of failure to deliver arbitrarily small by increasing the quit time.

We now examine the consequences of protocol failures in either the receiving discipline or the sending discipline. This analysis was suggested in an informal note by Belsnes (1974a).

THM 2: A PAR protocol that is functioning incorrectly because the received-packet ID list is lost (receiver crashes and restarts) will either lose packets, generate duplicate packets, or fail to deliver packets, and the failure probability cannot be made arbitrarily small by the sender.

PROOF: Suppose the protocol was initially functioning correctly. Let side A be sending packet X to side B.

Suppose that when B fails, it loses its received-packet ID list, but then continues to function normally. Suppose the original transmission of X arrived intact at B and was

delivered, but the ACK was damaged or delayed. Then B fails, clearing its received-packet ID list. A retransmission of X then arrives, and is not detected as a duplicate, hence is delivered to the process.

Alternatively, suppose that when B receives any packet from A after failing, it notifies A of the failure, and rejects any packets until the protocol is initialized again. In this case A reinitializes the protocol (by some foolproof means beyond the scope of this analysis). But then A must decide what to do about X:

If A sends X, it may be a duplicate as above.

If A doesn't send X, and reports success, the packet may be lost (if B failed before receiving a good copy of X).

If A notifies the process of the failure and the uncertain fate of X, the process has the same possibilities for failure:

Continue trying to send X which may result in a duplicate as above. (This couldn't happen in THM 1.)

Give up which may be a failure to deliver X. Furthermore, the sender cannot make the probability of failure to deliver arbitrarily small by changing

parameters available to him, since this failure depends on the reliability of the receiver.

THM 3: A PAR protocol that is functioning incorrectly because the sending discipline loses track of ID's used or packets pending (sender crashes and restarts), will either lose packets, fail to deliver packets, or force the sending process to duplicate packets.

PROOF:

LOSS: If the sender loses track of ID's, and reuses an ID for a new packet, the receiver will ACK it but discard the packet as a duplicate. However, the sender will receive the ACK and report successful delivery.

FAILURE TO DELIVER: If the sending discipline loses packets that have been transmitted, but not yet acknowledged, it ceases to retransmit them, and they are not delivered. Furthermore the process may not even be notified of the failure.

DUPLICATION: If the sending process tries to recover from the absence of either a success or quit notification from the protocol by resending a packet, the packet may have already been delivered before the failure.

Theorems 1-3 demonstrate the fundamental limitations of PAR protocols: they successfully mask errors in the transmission medium, but they cannot guarantee reliable transmission when part of the protocol itself is violated due to failure of one side or the other. The information maintained at both sides of the protocol is necessary for correct functioning.

Many protocol designers persist in trying to get around this fundamental limitation and "invisibly" recover from failures by introducing more complicated control mechanisms, usually involving reinitializing the connection [Mader74]. The issue of (re)initializing a connection for reliable transmission after a failure (cf section 5) is separable from the issue of reliability within a connection. Theorems 2 and 3 show that given certain types of failure, there can be no guaranteed reliability with PAR type communication protocols.

Those desiring greater reliability may implement failure recovery schemes at a higher (process) level (where they meet the same problems), or reduce the possibility of protocol failure with self checking or redundant machines, backup stores, checkpointing, or other means.

4. PAR PROTOCOL WITH SEQUENCING

The basic PAR protocol above does not concern itself with sequencing. When the characteristics of the transmission medium include out-of-order delivery (frequently the case in packet switching nets), the basic PAR protocol must be augmented with a sequencing mechanism if correctly sequenced interprocess communication is desired. This section incorporates such a mechanism into a PAR protocol, resulting in a Sequencing PAR or SPAR protocol. Delivering packets in order is now included in the protocol performance requirements.

Sequencing is normally achieved by including a sequence number (SN) in the control information attached to each packet by the sending discipline. The receiving discipline uses SN to determine the correct order of arriving packets. First we describe a SPAR protocol using both sequence number and unique identifier (cf section 3) fields in each packet. We show that the sequence number may also serve as a unique identifier, eliminating the need for a separate packet ID field. We then define a class of simplified SPAR protocols and analyze its reliability as in section 3.

DEF: A Sequencing Positive Acknowledgement, Retransmission (SPAR) protocol is a PAR protocol with the following additions:

SENDING DISCIPLINE: The sending discipline maintains a sequence number (SN). Each packet submitted by the process has SN attached (along with ID), and then SN is incremented.

RECEIVING DISCIPLINE: The receiving discipline maintains an expected sequence number (ESN). After discarding damaged packets, the packet's ID and SN determine the action to be taken according to Table 1.

Table 1

Processing of Received Packets in SPAR Protocol

	packet SN : ESN		
	lower	equal	higher
ID new	XXX	ACK, deliver to process, INC, ENTER	discard as out of order
ID old	ACK, discard	XXX	XXX

ACK means transmit an ACK referencing ID;
INC means increment ESN;
ENTER means enter the packet's ID in the received-packet ID list;
XXX means this case does not occur.

The protocol is initialized when SN and ESN are equal to each other (may be different in the two directions), no packets have been sent, and both sides have empty received-packet ID lists.

From Table 1 we see that the sequence number and identifier fields in a packet maintain redundant information--they are both duplicate suppressors. In particular, the receiving discipline never needs to check the received-packet ID list to detect duplicates, because the ESN screening accomplishes this. Since it is easier to remember a single ESN than a potentially infinite list of ID's, the ID can be dropped entirely from the SPAR protocol, with the sequence number performing both the duplicate detection and sequencing functions. The resulting simpler SPAR is specified as follows:

SENDING DISCIPLINE (see figure 4):

> The sending discipline maintains a sequence number (SN). Each packet submitted by the process has SN attached, and then SN is advanced to its successor. (1) The packet is transmitted, and a copy retained.
>
> Arriving ACK's are checked for errors, and damaged ones discarded.
>
> When an ACK referencing this packet's sequence number is received, the retained copy is discarded (and the sending process notified of success). If no ACK is received within the retransmission timeout period R, the copy is retransmitted and the cycle repeated. If the quit time has been exceeded, retransmission is suspended (and the sending process notified).
>
> ACK's for discarded packets are ignored.

---

(1) The simplest and most widely used successor function is to increment by one, although more complex successor relations have been used to support priority (IMP-IMP protocol [McQuillan72]) or fragmentation and reassembly [Cerf74b].

FIGURE 4 SPAR PROTOCOL SENDING DISCIPLINE

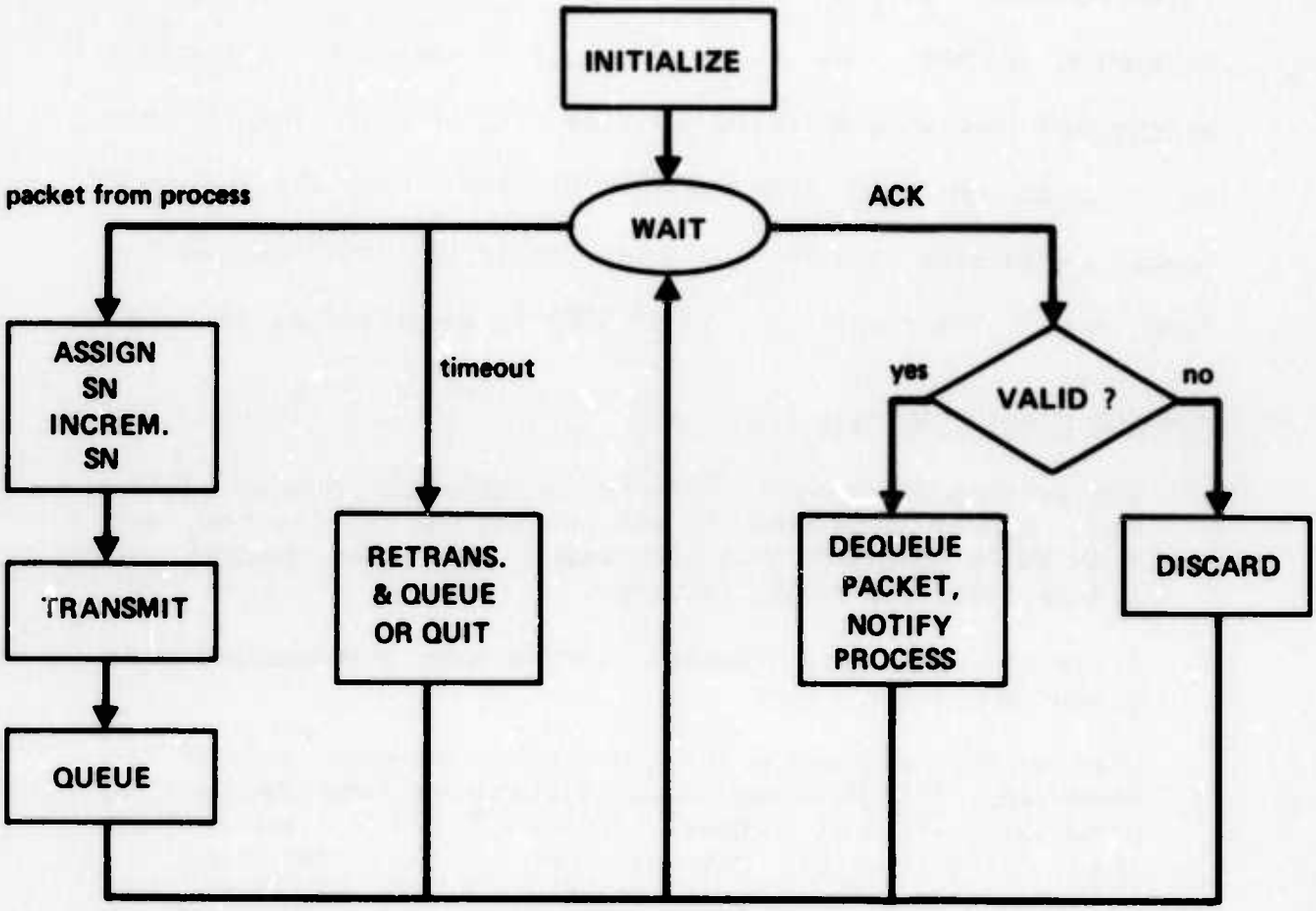
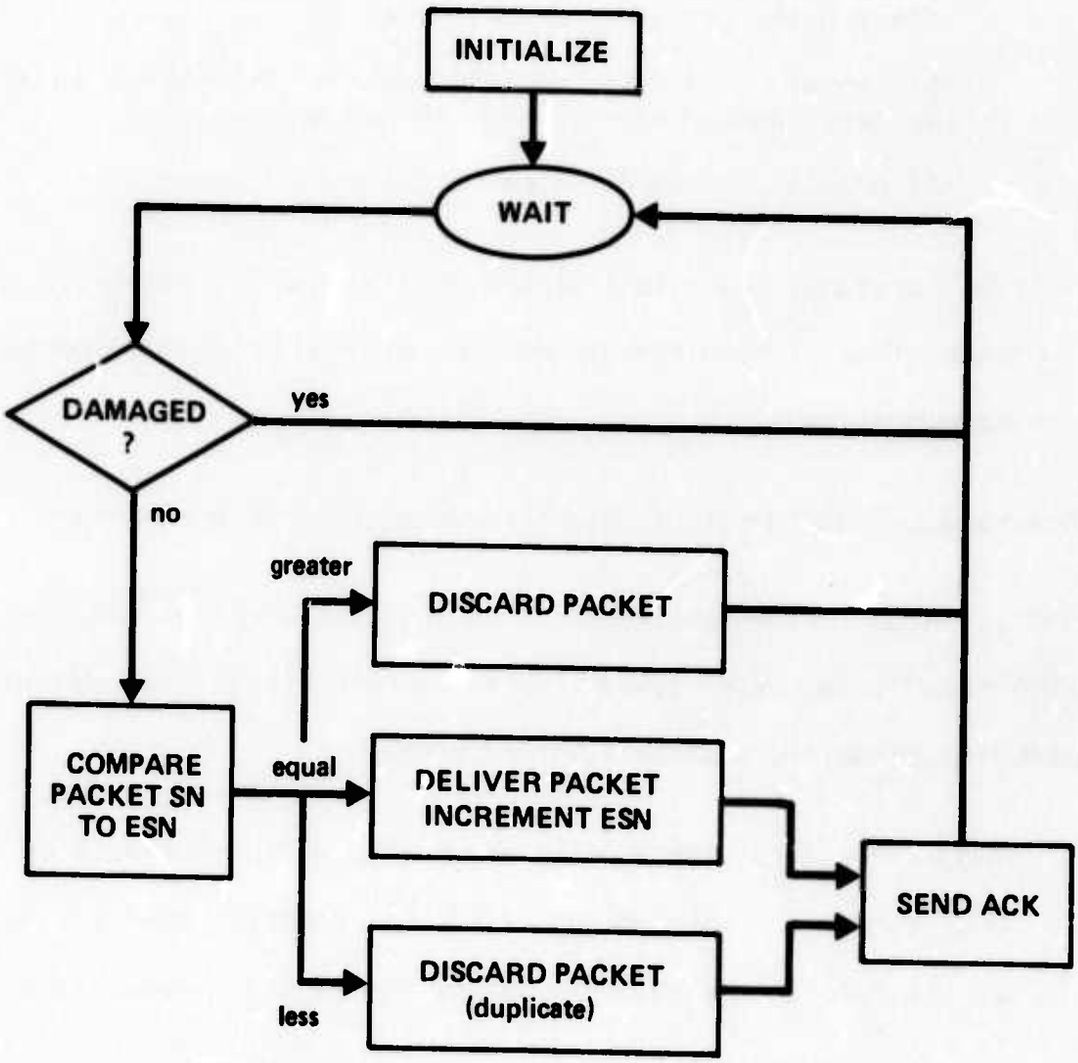


FIGURE 5 SPAR PROTOCOL RECEIVING DISCIPLINE



RECEIVING DISCIPLINE (see figure 5):

The receiving discipline maintains an expected sequence number (ESN).

Each packet received is checked for errors, and discarded if damaged.

If not damaged, the packet's sequence number (SN) is compared with ESN. The receiving discipline operates as follows on the basis of this comparison:

If less, transmit an ACK referencing the packet's SN and discard the packet as a duplicate.

If equal, transmit an ACK, deliver the packet to the process, and advance ESN to its successor.

If greater, discard the packet as out of order. (1)

The protocol is initialized when SN and ESN are equal to each other in both directions (see section 5) and no packets have been sent.

Theorems 1-3 carry over straightforwardly to SPAR protocols.

THM 1B: A correctly functioning SPAR protocol with infinite quit time never loses packets, duplicates packets, fails to deliver packets, or delivers packets out of order.

PROOF: The first three parts are proved as in theorem 1 with the sequence number acting as ID. If a packet ever arrives at the receiving discipline before one of its predecessors,

(1) For greater efficiency, the receiving discipline may choose to keep some number of out of order packets for a time. The costs and benefits of such schemes will be discussed in chapter III.

the ESN check will cause it to be discarded. Only the next packet in order can be delivered to the process.

THM 2A: A SPAR protocol that is functioning incorrectly because the receiving discipline loses ESN (receiver crashes and restarts), will either lose packets, duplicate packets, or fail to deliver packets.

PROOF: Same as theorem 2 with ESN taking the place of ID.

THM 2B: A malfunctioning SPAR protocol where ESN and SN become desynchronized may completely fail to deliver packets.

PROOF: Desynchronization may occur if either the sending or receiving discipline fails to maintain SN or ESN correctly. If ESN winds up below or above the sequence number of all outstanding packets (outside the "window" of expected sequence numbers described in [Cerf74b]), the "expected" sequence number will never appear at the receiving discipline, and no packet will be accepted. Recovering from such deadlocks requires resynchronizing the protocol as discussed in section 5.

Even if SN and ESN are lost or misset, a malfunctioning SPAR will not deliver packets out of order as long as the ESN screening in the receiving discipline is obeyed. (However, a series of in-order duplicates may be delivered as in THM 2A.)

Theorem 1B shows that SPAR protocols provide the desired reliability characteristics when functioning correctly. Under protocol failures, however, invisible error-free recovery is again impossible to guarantee, and SPAR protocol failures may even result in total deadlocks of the communication path.

Both the infinite sequence number space assumed in this section for SPAR protocols, and the infinite identifier space assumed for PAR protocols in section 3 are impossible in practice. For SPAR protocols, a finite sequence number space places constraints on the volume of traffic transmitted. If the maximum packet lifetime is L, no sequence number can be reused for time L, limiting the rate of transmission. If the size of the sequence number space is N, Cerf and Kahn (1974c) have shown that at most N/2 packets can be outstanding (transmitted but not yet acknowledged) at any time. A suitable modulo N successor function and comparison operations are also required.

If these constraints are violated, "old" packets with acceptable sequence numbers may appear at the receiving discipline and be accepted instead of the current packet with the same sequence number (cf section 5.2). These constraints must be included in the protocol specification in order to assure reliable operation.

Similar constraints on the reuse of packet identifiers by the sending discipline apply to (nonsequencing) PAR protocols. Maintenance of the received-packet ID list by the

receiving discipline presents other difficulties. Received identifiers must be removed from the list after time L so the next use of the ID will be accepted. This may adequately reduce the size of the list with low transmission rates or small L. Further reductions may be accomplished by assigning identifiers sequentially so that remembering a single ID can represent the fact that all previous ID's have been received. Only the relatively small number of noncontiguous ID's must be remembered individually. Sequencing also provides a simple means of generating unique identifiers at the sending discipline. Hence sequence numbers provide the cheapest way to keep track of packets already sent or received, even if the sequencing information is not used to deliver the packets in order. Pouzin (1974c) has described a combination bit map and sequencing mechanism to further reduce storage requirements.

5. CONNECTION ESTABLISHMENT

Sections 3 and 4 have focused on the operation of a communication protocol after the protocol is initialized. The analysis considered a single conversation between two processes desiring to communicate with each other. This section examines the additional issues involved in beginning and ending a conversation.

After clarifying the concept of a connection between processes for reliable communication, we discuss the actions required to establish a connection and show that some simple mechanisms proposed for this purpose are inadequate with a hostile transmission medium. We present more robust connection establishment mechanisms and demonstrate their correctness under normal operation and the consequences of various failures. Appendix A develops a state diagram model for representing connection establishment procedures which is used to analyze both simple and robust establishment mechanisms.

The need to consider explicitly starting and ending conversations arises for several reasons [Pouzin75]:

(1) In order to function correctly, the protocol must be initialized before a conversation begins.

(2) In reallty, many processes will want to communicate with many other processes. If there are N processes, there are

N(N-1)/2 possible conversations (assuming no one talks to himself), but the number of conversations actually active at any moment will generally be far smaller. Without a mechanism for starting and ending conversations on demand, the state of all possible conversations must be maintained perpetually at an impossible cost for even a moderate number of processes.

(3) In the case of certain protocol failures (Host crashes), the protocol must be reinitialiazed to allow reliable comunication to proceed from the time of fallure (see theorems 2-3).

(4) Processes may wish to make themselves available for communicaation at some times, and refuse conversation at other times.

5.1 Connection Definition

The notion of a conversation can be formallzed as follows: A connection is a bi-directional communication mechanism between two processes. A connection is uniquely specified by a pair of processes. That is, once the idea of multiple connections between various processes is introduced, the communication protocol must provide a means for identifying processes and hence connections. These process ID's are called

addresses, and a connection is specified by a pair of addresses. We denote connections by address pairs in angle brackets, <address, address>.

To start a conversation, two processes OPEN a connection, and to end the conversation they CLOSE the connection. This leads to the specification of states of a connection:

> ESTABLISHED: when the protocol has been initialized and the processes are free to exchange packets.
>
> NOT ACTIVE: when the protocol is not initialized and the processes do not intend to communicate. A minimum of state information about the connection is maintained.

The communication protocol attempts to establish a connection upon a process's request to OPEN a connection, and to terminate the connection on the process's command to CLOSE. An incarnation of a connection is the time from the establishment to the closing of the connection. A connection <A,B> may go through many incarnations as processes A and B open and close a communication path over time.

Without fully specifying the details, we name the new class of protocol that includes a mechanism for opening and closing connections a Communication Control Protocol (CCP). A CCP is a SPAR protocol with the additional mechanisms necessary to reliably initialize and terminate the protocol.

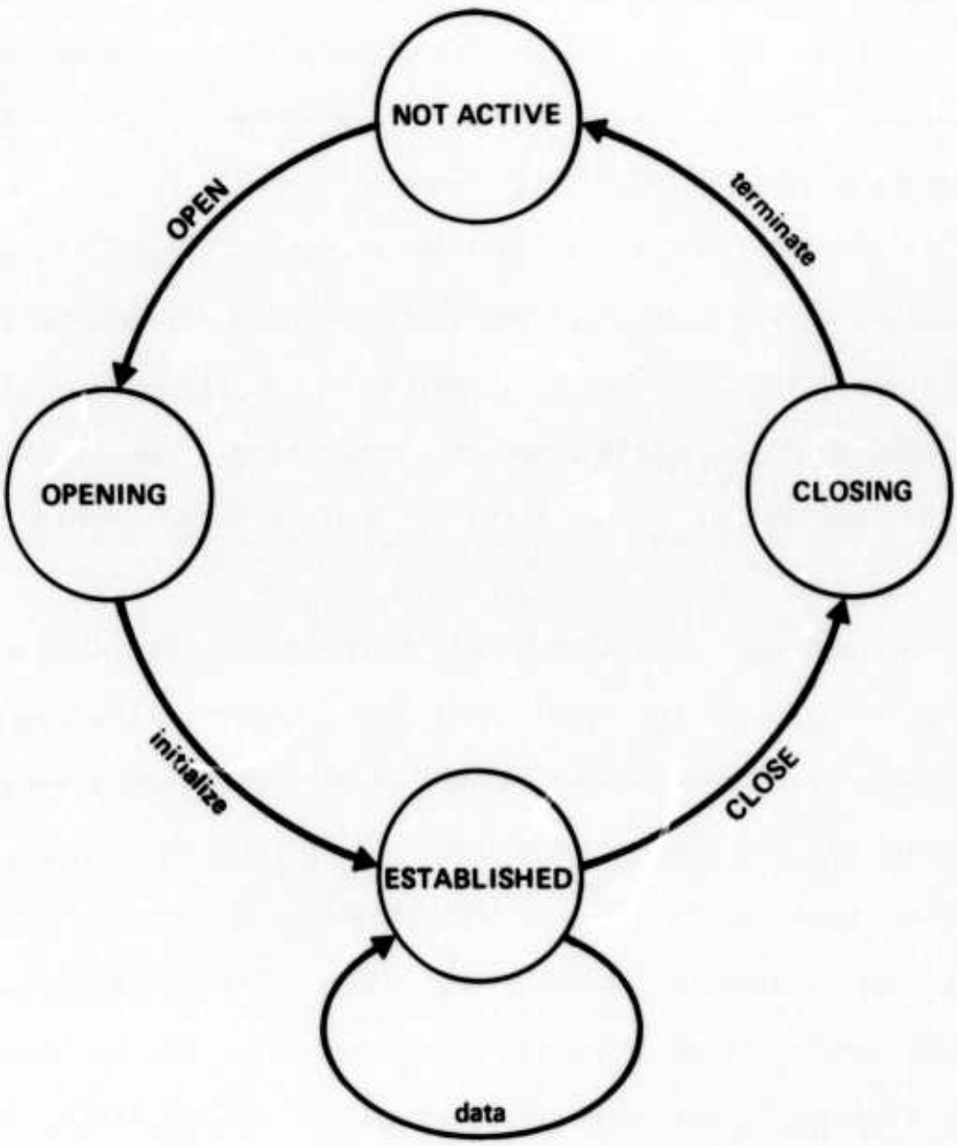
To move a connection from the Not Active state to the Established state, the protocol must be initialized, and the connection may spend some time in an intermediate state called OPENING. In going from the Established state to the Not Active state, the protocol should terminate communication in an orderly fashion (perhaps wait for outstanding packets to be received or acknowledged), and the connection may spend some time in an intermediate state CLOSING. (See figure 6)

It is important to note that the protocol disciplines on the two sides of the connection may think that the connection is in different states. The full state of a connection is specified by a pair of states, one for each side. We denote connection states by state pairs in angle brackets, <state, state>.

The correct functioning of a protocol can now be considered in terms of the state transitions. Each of the major states above may have a substructure of more detailed states. For example, the exchange of data packets described in section 4 occurs with both processes in the Established state. The analysis of possible transitions and determination of undesirable states is an extremely useful technique for protocol analysis as we shall see later in this section. But first we examine the means for opening and closing a connection.

In the simplest system, connections might be opened and closed by some means external to the communication system. For

FIGURE 6 STATES OF A CONNECTION BETWEEN TWO PROCESSES



example users might call each other up, or physically move from one place to another and instruct the CCP to initialize or terminate a connection. Such systems will be called externally controlled. The coordination of the two sides is enforced externally by some higher authority, subject to its own validation problems.

However, external control is frequently not possible or desirable. The most interesting and useful systems use the transmission medium itself to control connections as well as to communicate processes' data. To this end, CCP's exchange control packets. Only such internally controlled systems will be considered further, although the pitfalls discussed below apply to externally controlled systems as well.

5.2 Opening a Connection

Suppose for concreteness that processes A and B wish to open a connection. The primary task in opening the connection <A,B> is is to initialize the protocol. Each CCP has SN in the sending discipline, and ESN in the receiving discipline as described in section 4.

1) SN(A) and SN(B) must be set to some initial values, ISN(A) and ISN(B).

2) ESN(A) must be set to ISN(B) and ESN(B) must be set to ISN(A).

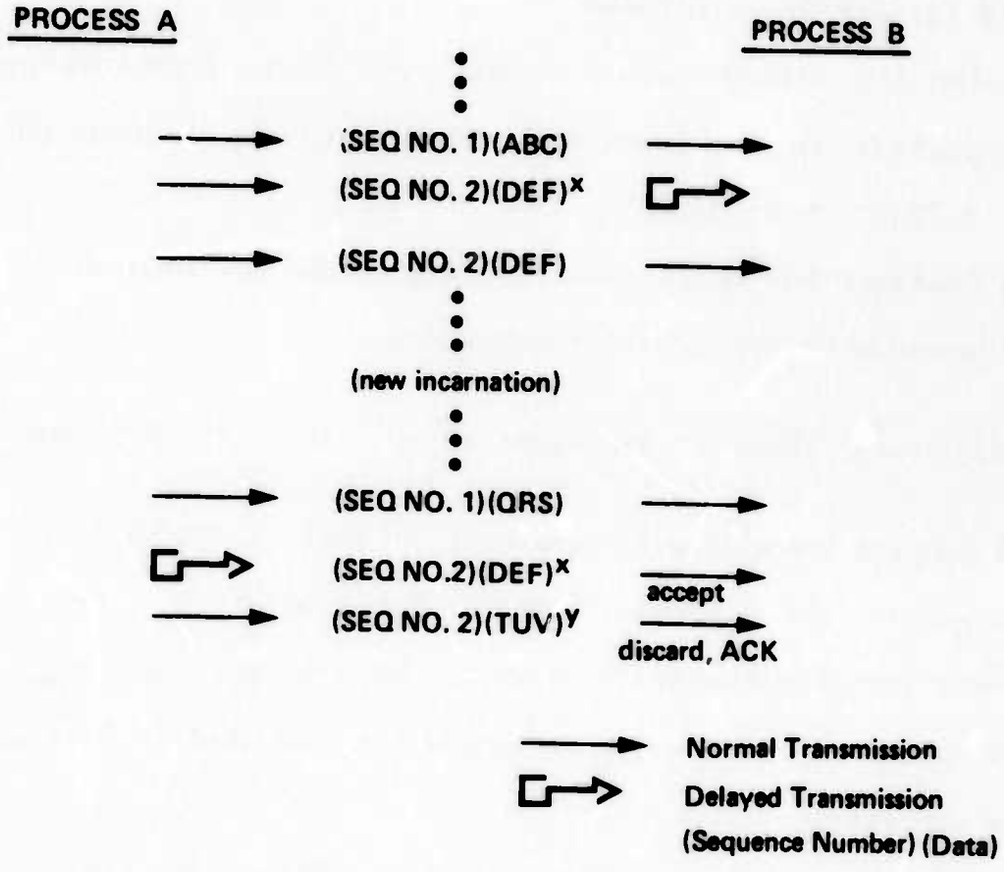
5.2.1 Selecting ISN

The conditions for initialization of SPAR protocols required that there be no packets exchanged between A and B. This may not be true if A and B have been previously connected, and in fact packets from the previous incarnation of a connection may emerge, due to delays in the transmission medium and out-of-order delivery, during the current incarnation. The sequencing mechanism defined for SPAR protocols successfully handles duplicates within a single connection, but cannot in its simple form reliably manage opening and closing connections. In particular if ISN is picked for the new incarnation so that some sequence numbers from an old incarnation are reused, errors may occur.

THM 4: A CCP that transmits packets undifferentiable as to connection incarnation (by reusing sequence numbers) will lose packets, duplicate packets, and deliver packets out of order.

PROOF: (see figure 7) Suppose packet X from an old incarnation of connection <A,B> and packet Y from the current incarnation of <A,B> are assigned the same sequence number by A and transmitted to B. Furthermore suppose X was retransmitted during the old incarnation. If the

FIGURE 7 ERROR DUE TO REUSE OF SEQUENCE NUMBERS



(retransmitted) packet X arrives at B before Y, it may be accepted and acknowledged in place of Y, and packet Y will be discarded as a duplicate. A will receive the ACK and think Y was successfully transmitted. Message X is duplicated and delivered out of order, while message Y is lost.

This failure occurs because:

(1) The transmission medium can delay or store (retransmitted) packets so they reach their destination out-of-order during a later incarnation.

(2) Messages from old incarnations may not be distinguished from packets of the current incarnation.

Accordingly, there are two types of solution to the problem:

(1) Suppose there is a maximum time L that a packet can be stored in the transmission medium (see chapter I). Then if no connection is opened before time L after its last closing, all old packets will be gone, and any ISN may be used to initialize the connection.

This solution requires CCP's to remember for time L that a connection was closed, and hence runs counter to the goal of minimizing state information maintained for Not Active connections. Furthermore, if a Host fails and forgets which connections were recently closed, it must prevent opening

connections for ALL its processes for time L since any of them might have had recently closed connections. The cost of this type solution is then storage of status information, and delay in reestablishing connections after failures. When L is large, these costs may be high. A recent proposal to CCITT for an international standard of 30 seconds for L makes this approach more attractive [INWG75].

(2) Be sure packets from the current incarnation can be distinguished from old packets. Ways to achieve this second type of solution include:

(a) Set ISN to the last sequence number from the previous connection. This also violates the absence of state information for inactive connections because the last sequence number used must be remembered for time L on every connection. Once time L has passed, any value for ISN may be used. If a Host fails, all connections must wait time L as in type 1 solutions.

(b) Set ISN from a single clock for all connections at a Host [Tomlinson74]. The clock value is the only state information that must be preserved through inactive connections and host crashes. This scheme requires resetting the sequence number (resynch) if the clock cycles around to where the sequence number is. The time until

resynch is required is determined by the sequence number field size, clock rate, and connection traffic intensity [Dalal74]. An additional cost of this mechanism is these resynch tests.

(c) Add more identifying information to each packet so otherwise identical sequence numbers can be distinguished. This requires keeping an "incarnation number" for each connection, or possibly a global single ID which is assigned to each new incarnation, and then incremented. If the ID has cycle time greater than L, no confusion is possible. For the single global ID, only a single number need be remembered for all connections as in (b). Another field on every packet sent is required, increasing overhead.

In general, all solutions of type 2 may fail if the state information which distinguishes previous incarnations is lost. In this case the CCP must resort to a type 1 solutions as shown in theorem 5 below. To reduce the likelihood of failure, the state information can be reduced to a minimum and maintained by some specially reliable mechanism like an independent clock or counter.

THM 5: A CCP with finite maximum packet lifetime L that fails by forgetting the state of connections must either inhibit all transmission for time L after the failure, or will lose packets,

duplicate packets, deliver packets out of order, and fail to correctly initialize connections.

PROOF: When the CCP forgets the state of a connection, it loses whatever state information is used to differentiate packets from different incarnations of the connection as described above. Then it may restart by resetting this state information to a value used earlier, introducing packets in the current incarnation that are undifferentiable from packets of a past incarnation. Then by theorem 4, packets may be duplicated or delivered out of order. In particular, the control packets causing initialization (discussed in the next section) may be lost or delivered out of order, causing incorrect initialization. To avoid these problems, the CCP must wait time L after a failure before transmitting any packets.

Theorems 4 and 5 extend the results of theorems 2-3 to CCP's. Loss of state information allows new packets to be transmitted on a connection when it is still possible for old (retransmitted) packets that look the same to arrive at the receiving discipline and be accepted instead. In practice a combination of minimizing the possibility of state information loss and waiting some time after restarts may reduce the probability of confusion to an acceptably low level. Transmission media that guarantee in-order delivery avoid this problem.

Confusing packets from different incarnations of a connection is not as unlikely a problem as might be supposed. Many protocols use zero for the initial sequence number every time a connection is established. In these protocols it is quite possible to open a connection, close it, and reopen it within a maximum packet lifetime L. In this event it is quite likely that retransmissions from a previous incarnation will emerge with correct sequence numbers to be accepted during the current incarnation.

The worst difficulties occur when the control packet(s) that initialize a connection get confused. Then one or both CCP's may think the connection is established, but SN is not equal to ESN and no packets can be successfully transmitted. A deadlock occurs which must be broken by further control message exchanges as described below, or by some external means.

5.2.2 Setting ESN equal to ISN

Once ISN is selected for a new incarnation of a connection, ESN must be set equal to ISN in both directions. To accomplish this, each CCP may try to keep the same state information that the other CCP uses to select ISN. This is not always possible, and where it is possible, requires a lot of work to synchronize all clocks, remember incarnation numbers for all Hosts, or remember old sequence numbers for all old

connections. And all this information must be set somehow in the "beginning" or after a failure.

This suggests that to establish a connection, the sending discipline should transmit a synchronization control packet (SYN) to the receiving discipline giving the value of ISN (see figure 8). The receiving discipline can set ESN to this value without maintaining any state information about its partner CCP. The receiving CCP returns a SYN giving its own ISN, or can reject any SYN that arrives when the protocol is in an inappropriate state (the only appropriate state is the Opening state where the process has signified its readiness to converse, but the connection is not yet established.) Inappropriately timed arrivals are either old retransmissions, protocol errors, or attempts to establish a conversation with an unwilling partner.

Unfortunately, this simple system of a credulous CCP is inadequate when packets may arrive out-of-order as shown by theorem 6 below. Once sequencing is initialized, sequence numbers serve to validate incoming packets. But while the connection is being initialized, there is no way for the receiving discipline to validate an arriving SYN since it maintains no state information about the other side's ISN.

THM 6: A CCP that maintains no state information about ISN for the remote end of the connection, but accepts ISN from an

FIGURE 8 SIMPLE CONNECTION ESTABLISHMENT USING SYN CONTROL PACKET

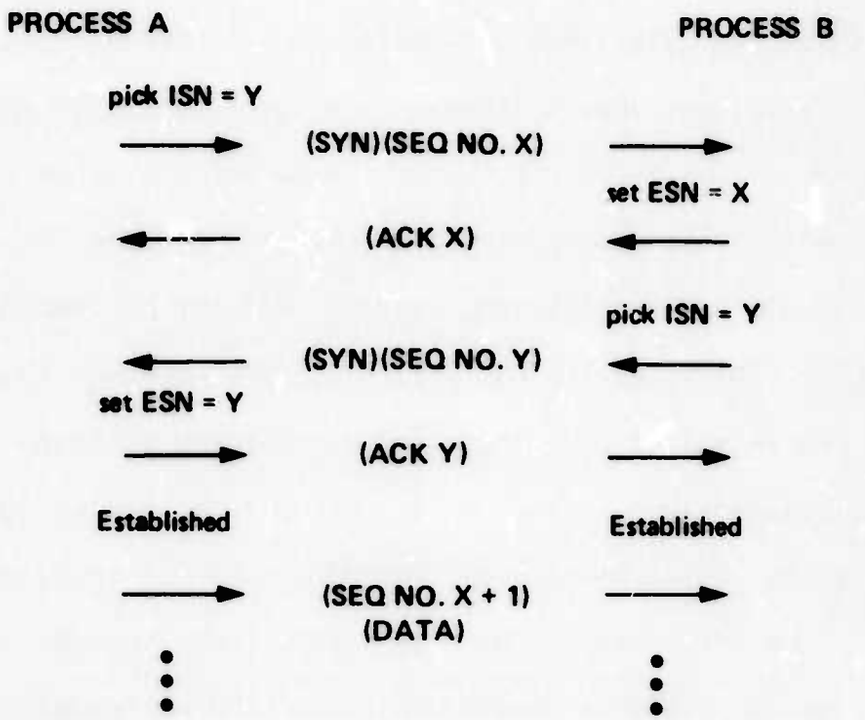
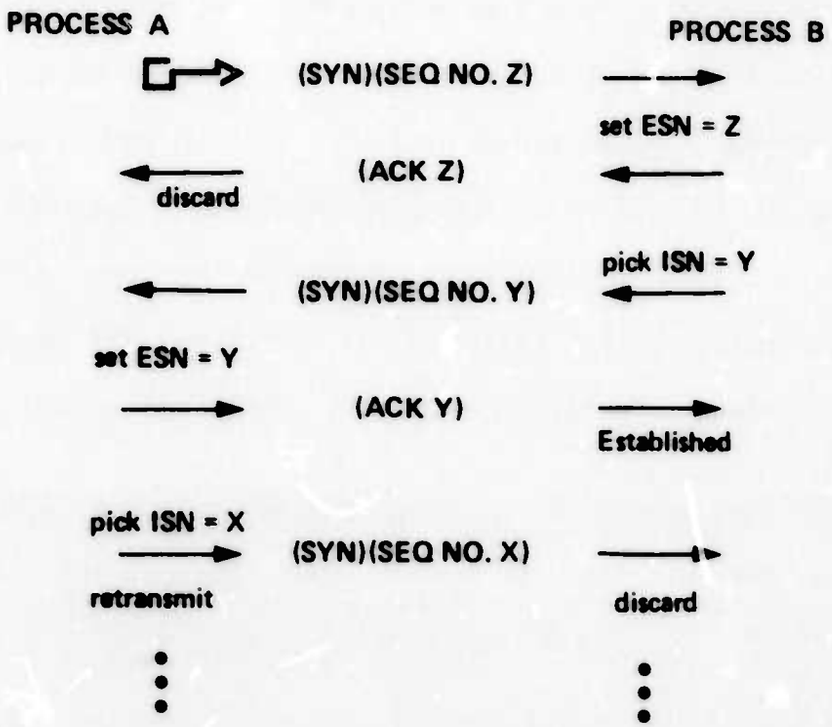


FIGURE 9 SIMPLE CONNECTION ESTABLISHMENT ERROR



arriving SYN control packet, may incorrectly synchronize the connection and cause a deadlock.

PROOF: (See figure 9) Suppose a SYN control packet with an old ISN was retransmitted and delayed in the transmission medium during a previous incarnation of the connection. This old SYN may arrive just when the new connection is in the opening state, and be accepted as valid. ESN will be set to the old ISN, and the connection state set to Established. But no data will be accepted as in theorem 2C.

"3 Way Handshake"

To avoid this problem, a more reliable means of transmitting the current ISN to a CCP must be used. Tomlinson (1974) has presented such a scheme called the "3 way handshake." Instead of simply accepting an arriving SYN, the receiving CCP must ask the sending CCP to verify the SYN as current. The receiving CCP returns a SYN-Verify control packet to the sending CCP which refers to the ISN from the SYN (see figure 10). If the SYN was a current packet, the sender returns a positive acknowledgement (ACK), and only then does the receiver accept the SYN and set ESN. This synchronization must occur in both directions, with the SYN-Verify also carrying ISN of the receiver in the other direction.

If the SYN-Verify references an old ISN (See figure 11), the sender returns a negative acknowledgement (NACK) and the

FIGURE 10 "3 WAY HANDSHAKE" CONNECTION ESTABLISHMENT

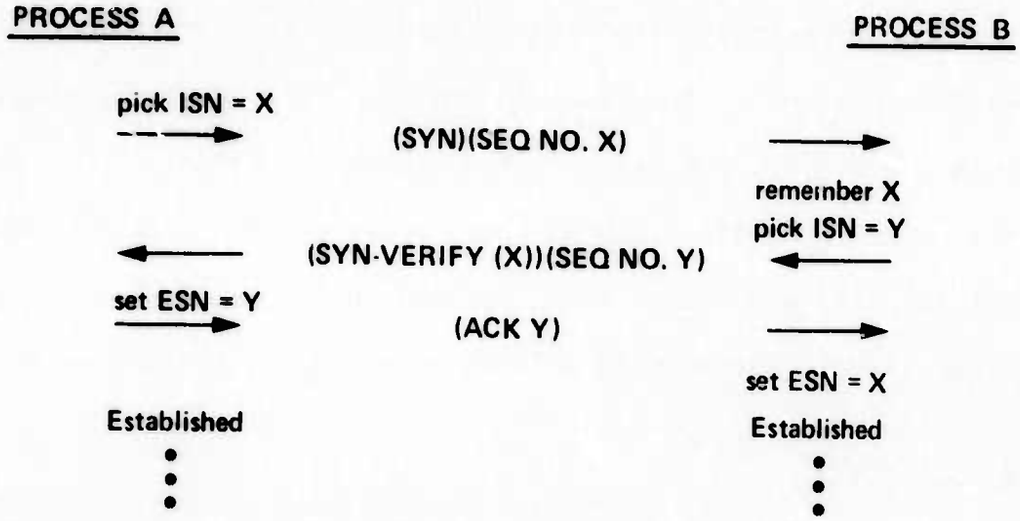
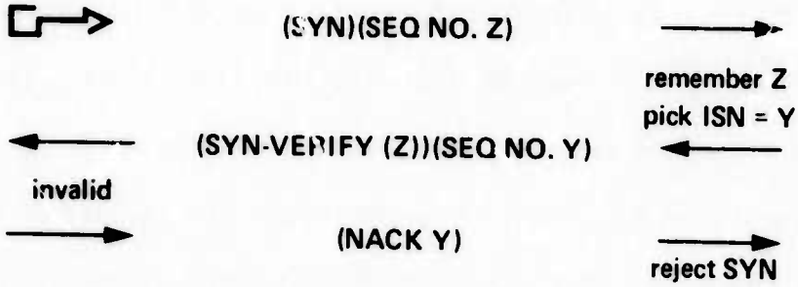


FIGURE 11 REJECTION OF OLD SYN PACKET WITH "3 WAY HANDSHAKE"



receiver discards the SYN. This takes care of old (retransmitted) SYN or SYN-Verify packets.

Collision Avoidance

This "3 way handshake" mechanism for establishing connections is inherently asymetric, with one side initiating the attempt by sending a SYN, and the other side waiting to respond to a SYN from the active side. However, some processes may not have agreed on an active and passive side and both sides may attempt to initialize the connection. Then each side will see a simple SYN rather than a SYN-Verify in response to its own SYN. In this case a collision is said to occur, and the collision resolution mechanism used in broadcast transmission media [Abramson73a] may be applied. Both sides "forget" that they have sent or received any SYN's and wait a random amount of time before trying to initialize the connection again.

Several authors have investigated the relationship between retry intervals, propagation time, and time until success in broadcast media [Metcalfe73, Abramson73a]. If the retry time distribution is wide relative to the propagation delay, then very likely one side will try again and have its SYN delivered while the other side is still waiting, avoiding a second collision. The collision avoidance mechanism simplifies connection establishment since it reduces simultaneous initiations to the more tractable one-sided attempts. This

application of collision avoidance to connection establishment is believed to be a new technique.

In order to reduce the frequency of collisions, Tomlinson has suggested that a CCP enter a special simultaneous initialization state when it detects a collision [private communication]. Dalal (1975) has developed algorithms which allow the connection to be reliably established for "normal" but simultaneous initialization attempts. However, if an "old" SYN from a previous incarnation arrives during a simultaneous initialization attempt, the CCP must still give up and retry from scratch.

5.2.3 Correctness of Connection Establishment Mechanisms

Sections 5.1 and 5.2 have shown the shortcomings of some simple protocol initialization mechanisms, and suggested more complicated mechanisms to successfully deal with transmission medium characteristics. In this section we prove that a correctly functioning CCP using the ISN selection and 3 way handshake mechanisms described above does indeed correctly establish connections for reliable interprocess communication.

THM 7: A correctly functioning CCP (with infinite quit time) using ISN selection and 3 way handshake mechanisms above, correctly establishes connections despite transmission medium

characteristics of loss, delay, damage, duplication, and out-of-order delivery of packets.

The proof of theorem 7 is based on a state diagram model of the protocol process on each side of the connection. These two processes interact by exchanging packets which we assume may be lost, duplicated, or delivered out-of-order since we are particularly interested in developing robust protocols for worst case situations. Each protocol process is driven by events including user commands, packet arrivals, and internal timers.

The complete state of the system includes both protocol processes' states and the packets in the transmission medium. A large reduction in complexity is achieved by classifying all packets in the transmission medium as either "current" packets or "old" packets (cf appendix A). Only current packets must be explicitly represented as part of the composite state.

Appendix A proves theorem 7 and also reproves theorem 6 using the composite state formalism to show the correctness of a powerful protocol and the inadequacy of a simple protocol for connection establishment. Failure recovery techniques are also considered.

5.3 Closing a Connection

The purpose of closing a connection is to return the connectior to the Not Active state, freeing the resources associated with maintaining the connection. The tables, buffers, and other data structures used to support the connection are then available for other connections.

The CLOSE command means that the process does not want to send or receive any more packets. Normally processes exchange data signaling the end of their conversation, and then request the CCP to close the connection. In this case when processes on both sides of the connection request termination, the CCP at each side can simply return all resources and place the connection in the Not Active state without exchanging any control packets. This simple scheme relies on both processes cooperating to close the connection.

However, some processes may not have an agreed termination procedure, or one process may wish to terminate the connection while the other attempts to continue. The simple unilateral termination scheme above might leave the connection with one side in the Not Active state, while the other side thinks the connection is still Established and continues to use transmission medium resources in useless (re)transmissions.

Since successful communication requires cooperation by both sides, when either side attempts termination, both sides

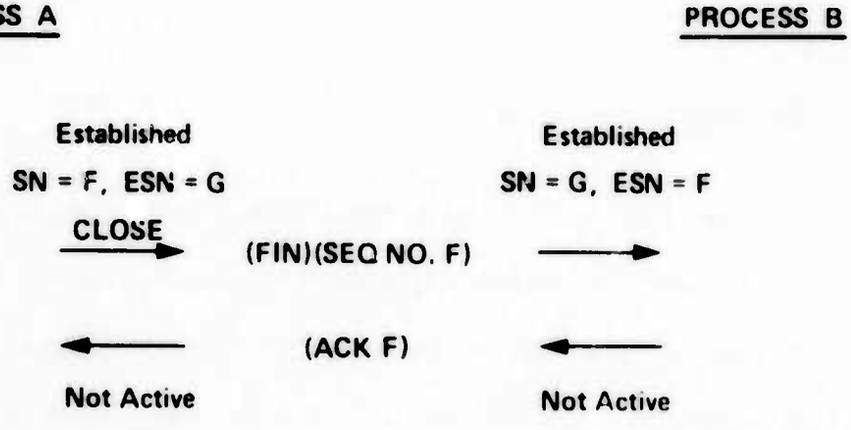
should be closed. To accomplish this, when the CCP gets a CLOSE request, it creates and transmits a termination control packet (FIN). This control packet is easier to validate than the initialization control packets discussed in section 5.2 because the connection is already established.

5.3.1 FIN Mechanism (See figure 12)

The sending side places the normal next sequence number in the FIN, and the receiving side uses the sequence number to determine whether the FIN is valid or an old duplicate just as for data packets. Furthermore, the sequence number determines exactly where in the data stream the FIN occurs, so that the receiver can wait for any outstanding packets if the FIN has arrived out of order. The receiving discipline returns an ACK for the FIN just as for data packets. It then notifies the process that the other side has terminated the connection, and places the connection in the Not Active state.

When the sending discipline sees the ACK for its FIN, it knows that the other side has terminated the connection and it can finish closing the connection on its own side. In this way when either process closes the connection, the resources at both sides are freed, and the state of the connection is kept consistent at both sides without depending on process cooperation. This mechanism allows both for cooperating

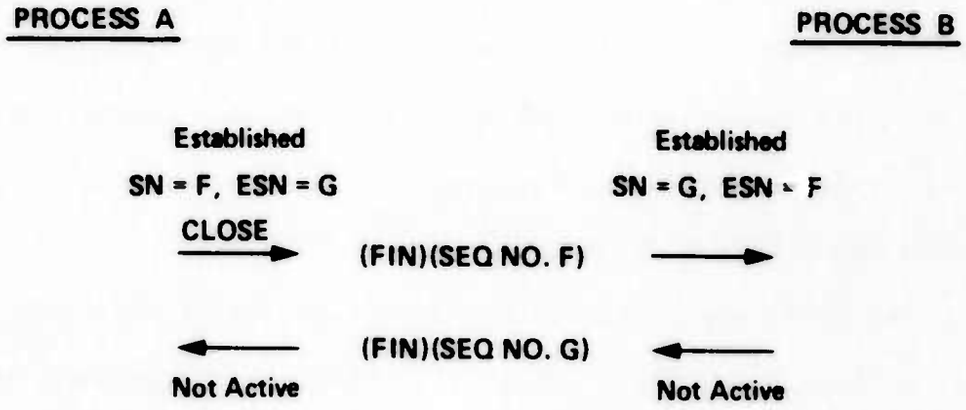
FIGURE 12 CONNECTION CLOSED WITH FIN CONTROL PACKET



processes to terminate the connection in an orderly fashion (after exchanging all desired data), and for one process to shut off the other uncooperative process and prevent useless activity.

To handle the case where both sides try to close the connection and send FIN simultaneously, the mechanism used in the ARPA net Host-Host protocol may be adopted [Carr70]. Instead of acknowledging a FIN with a normal ACK, the reply to a FIN is another FIN. Then the initiating and replying termination control mesages are identical, and simultaneous closes look like responses to both sides (see figure 13).

FIGURE 13 CONNECTION CLOSED WITH SIMULTANEOUS FIN PACKETS



5.3.2 Possibility of "Hung" Connections

Even with a FIN mechanism, a limited type of connection state inconsistency is still possible in closing a connection. To discuss this problem, we use the following notation: In closing a connection, both sides must move from the Established state (E) to the Not Active state (N) by passing through the Closing state(s) (C). The possible connection states are then <E,E>, <E,C>, <E,N>, <C,C>, <C,N>, and <N,N> counting symmetric states only once.

While the connection is in the Established or Closing states, the normal retransmission/duplicate detection mechanism using sequence numbers masks the effect of loss, reordering, and delay in the transmission medium. Once either side of the connection reaches the Not Active state however, essentially all information about the connection is lost and arriving packets are simply discarded (except SYN to start a new connection). This is exactly what is desired in the <N,N> state, but deadlocks are possible in the <E,N> and <C,N> states.

The unilateral close mechanism allows the <E,N> state to persist without any failure in the transmission medium, but because the processes fail to agree on closing the connection. This can be avoided by requiring the exchange of FIN control packets as described above.

The FIN scheme prevents the <E,N> state, but results in the <C,N> state if the ACK of a FIN is lost in the transmission medium. In this case, retransmissions of the FIN from the Closing side are discarded because the connection has already been inactivated. It is appealing to try to solve the problem by introducing another stage in the control packet exchange where the respondent to the FIN returns a FIN-Reply control packet and does not inactivate the connection until receiving an ACK for the FIN-Reply. Unfortunately this only shifts the problem to the other side and the final ACK.

THM 8: Any mechanism for closing connections in an internally controlled CCP allows either <E,N> or <C,N> states to occur where the connection will not terminate (enter the <N,N> state) using the normal closing mechanism.

PROOF: For unilateral termination schemes the <E,N> state can persist if one of the processes does not close its side as discussed above. For schemes involving exchange of FIN control packets, the <C,N> state occurs when the C side has sent the FIN type packet, and the N side has received this packet and returned an ACK. If the ACK is lost or damaged, the C side retransmits the FIN, but the N side discards the retransmissions because the connection is Not Active.

As noted in Appendix A, such "hung" or "half open" connections can also result from protocol <u>failures</u> where one side of the connection must restart in the Not Active state. To avoid such hung connections while closing a connection, three types of solution exist:

(1) A timeout mechanism whereby one side of the connection unilaterally "gives up" and goes to the Not Active state when it gets tired of waiting. This can be explicitly requested by the process (a sort of Reset command) or automatically performed by the CCP. This corresponds to the Quit time defined in section 2.

(2) A CCP in the Not Active state returns some special control packet when it receives a packet for a connection it considers inactive. The Connection Inactive control packet is a kind of negative ACK and refers to the sequence number of the arriving normal packet so the CCP that receives the NACK can verify that it refers to a current packet. Of course error packets are not returned for error packets. When a CCP in the E or C state gets a NACK instead of the expected ACK, it can close the connection. This corresponds to the Reject mechanism added to the protocol for failure recovery in Appendix A.

Another similar solution for a CCP in the N state that receives a FIN type control packet is to construct the appropriate ACK for return as if the connection were still active. This avoids special processing by the sender in the E or C state by shifting it to the receiver in the N state. This is not always possible since connection state information is generally discarded when the connection enters the N state, and the protocol may not know how to construct an appropriate ACK.

(3) When the CCP sends the final ACK before setting the connection Not Active, it can send n copies of the ACK, where n is large enough to "guarantee" that at least one will get through. This brute force approach is actually

used in at least one protocol known to the author (an early version of the ether net protocol at Xerox PARC used n=10).

5.4 Reducing Costs of a CCP

It is apparent from the above discussion that transmission medium characteristics of delayed out-of-order delivery of (retransmitted) packets cause difficult problems for reliable communication. One seemingly attractive approach to this problem is to require a transmission medium that delivers packets in order, or to implement a "low level" protocol mechanism that orders packets on a Host-Host basis, creating a first level virtual transmission medium that delivers packets in order, and simplifies the interprocess protocol design.

The direct cost of this approach is the cost of the sequencing mechanism itself with its own initialization problems. Where several connections share the same Host-Host sequencing mechanism, significant savings may result. When most connections are to different hosts, the two level mechanism, each level requiring independent state information and initialization, may result in increased delay and higher cost.

The indirect cost of this aproach is the interference between different connections now sharing the same ordering mechanism. When a packet from one connection is lost or delayed, subsequent packets on other connections cannot be

delivered even though they would be in-order on their own connection. If packet loss recovery mechanisms are also shared, then buffering constraints mean that a sluggish process that is slow to accept its arriving packets may hold up all the other processes sharing the same sequencing and error correction channel.

One disadvantage of a CCP is the relatively large overhead in packet headers and control packet exchanges required. This cost is particularly heavy for short single transaction applications [Kleinrock74]. Nevertheless, we have shown that given the hostile transmission medium characteristics described in chapter I, powerful mechanisms are necessary to guarantee reliable communication. A partial solution for transaction traffic may be to multiplex many transactions over a single longer duration connection. This introduces the interference between transactions mentioned above, but may be justified by savings in overhead and connection set up activity.

Chapter III

PROTOCOL EFFICIENCY

1. INTRODUCTION

This chapter considers the efficiency of interprocess communication protocols for computer networks. As with the reliability performance discussed in chapter II, quantitative performance delivered to processes by a communication protocol must be based upon the performance of the transmission medium underlying the protocol. Transmission medium characteristics most important to efficiency are delay, bandwidth, maximum packet size, and error characteristics.

To provide efficient interprocess communication based on these transmission medium characteristics, a protocol can attempt to optimize several internal parameters such as retransmission interval, packet size, flow control strategy, buffering, and acknowledgement scheme. Of course much of the performance seen by a process on one side is controlled by the behavior of the other process with which it is communicating. For example a protocol cannot on the average provide throughput to a source process that is greater than the acceptance rate at the receiver. In general, the maximum performance possible under ideal process behavior is of interest, as well as reduced performance due to limiting process behavior on one or both sides.

This chapter develops models to analyze the efficiency of successively more complex protocols. The two main performance measures chosen for analysis are average maximum throughput and mean delay since these represent the performance of primary interest to processes using the protocol. By throughput we mean the transmission rate of useful data between processes, excluding any control information or retransmissions that the protocol requires. By delay we mean the time from starting to transmit a packet at the sender to successful arrival of the entire packet at the receiver, or arrival of an acknowledgement at the sender in the case of roundtrip delay. We return to further define these performance measures later in this section.

Other efficiency performance measures of interest include retransmission rate, line efficiency, and buffer requirements. Retransmission rate indicates the number of times each packet must be transmitted and is a useful cost measure since packet communication costs typically include a per packet charge. Line efficiency is the ratio of useful traffic (throughput) to total traffic generated by a protocol including control information and retransmissions. It provides a measure of the overall efficiency of a protocol by indicating the fraction of total traffic that represents useful data. Buffers are required at the sender to hold packets until acknowledged, and at the receiver to hold packets until processed or for

sequencing out-of-order arrivals. Limited buffer storage restricts throughput and delay achievable.

Several authors have analyzed the efficiency of communication protocols for simple transmission media with fixed delay and no packet loss or reordering [Benice64a, Benice64b, Danthine75c, Pouzin73a, Burton72, Sastry74]. This study emphasizes performance analysis of protocols for interprocess communication over packet switching nets (PSN) with more complex and hostile transmission characteristics (cf section I-2).

Delay includes packet transmission time, or the time required to transmit all bits of a packet into the transmission medium (a function of the transmission medium bandwidth), and propagation delay, or the time required for a bit to travel from source to destination through the transmission medium. In a store-and-forward PSN, the propagation delay may itself have several components (cf section I-2) which we do not consider further.

Frequently it is important for the sender to receive a positive acknowledgement that the packet was delivered, in which case the roundtrip delay or time for successful delivery and return of response is the significant measure. A transaction system is an example of such a situation. Roundtrip delay includes delay for a packet to reach the receiver, processing time at the receiver, and delay for the response to reach the sender.

If a sending process produces packets for transmission at a high rate, the protocol may be unable to transmit packets immediately as they are submitted. In this case, submitted packets must be queued until they can be transmitted. The total delay seen by the process will consist of the waiting time while the packet is queued for service plus the normal delay to successfully transmit the packet through the transmission medium. During heavy demand periods, the total time to complete a requested transmission may be dominated by the waiting time. Under such conditions, the throughput is also important in determining total completion time because it determines the rate at which the waiting queue is emptied. To separate these effects, we explicitly exclude the above waiting time from our definition of delay.

While we define delay as an inherently single packet phenomenon, throughput concerns performance for a stream of packets. With simple protocols that transmit a single packet and then wait for its acknowledgement, throughput is simply the inverse of roundtrip delay multiplied by the useful bits per packet. By taking advantage of the pipeline or multi-server capacity of the transmission medium, a protocol can transmit multiple packets while waiting for acknowledgements and achieve higher throughput. The extent of this multiplexing is limited by transmission medium capacity, flow control mechanisms, and other constraints discussed in this chapter.

Achievable throughput is found to depend on six main factors: overhead, retransmission or error recovery, flow control or multiplexing, buffer allocation, receiver rate, and transmission medium bandwidth. These factors in turn depend on both protocol parameter settings and transmission medium characteristics.

For a simple PAR protocol with deterministic transmission delay on a single hop transmission line, Metcalfe (1973) has evaluated several of these factors. Section 3 extends the analysis of the error factor to include more realistic transmission delay functions for packet switching networks, and to include the effect of varying retransmission intervals. Section 4 considers the effect of flow control mechanisms on the multiplexing factor. In section 5 we discuss several acknowledgement, retransmission, and buffer allocation strategies, and consider the throughput degradation resulting from buffer limitations. Section 6 examines the effects of requiring sequencing at the destination (SPAR protocols). Section 7 briefly discusses the impact of packet size on protocol performance.

Effective delay depends more directly on the transmission medium characteristics, but packet size, retransmission interval, and sequencing requirements also have important effects. In general, minimum delay and maximum throughput are conflicting goals[Crowther75, Opderbeck74], so

protocol parameters must be adjusted to provide the type of service desired by a particular process (cf section 7).

In developing our efficiency analysis, we define a large number of parameters and performance measures in this chapter. To aid in remembering them, table 1 provides a list of names and brief definitions for the more important terms along with page numbers where they are first defined or discussed.

Table 1

Important Names and Variables Used in Chapter III

Name	Definition	Page
f(t)	Transmission delay distribution	83
F(t)	Transmission delay cumulative distribution	87
P	Packet length	83
B	Bandwidth of transmission medium	83
Tprop	Propagation time	83
P/B	Packet transmission time	83
LS	Loss or damage probability	83
TPmax	Average maximum throughput (useful data rate)	85
R	Retransmission interval	86
H	Header length	86
D	Data length in a packet	86
OH	Overhead	86
TPoh	Overhead factor in throughput	87
g(t)	Successful transmission delay distribution	87
G(t)	Successful trans. delay cumulative distribution	87
DL	Mean delay until successful delivery of a packet	89
Ntrans	Mean number of transmissions	89
TPretrans	Retransmission factor in throughput	90

Table 1 (cont'd)

Name	Definition	Page
Nwin	Window size for flow control	109
Tlocal	Packet transmission time (Host to Packet Switch)	112
Tnet	Roundtrip time through network less Tlocal	112
RHO	Ratio of service times or rates	112
UT	Utilization of sender	113
Nwinmax	Window size allowing maximum throughput	115
Nbuf	Number of buffers at receiver	123
Pfull	Probability that all buffers are full	125
TPbuf	Buffer limitation factor in throughput	127
Tint	Time between new packet transmissions	130
H(t)	Cumulative delay distribution with sequencing	131
DLseq	Mean delay including sequencing	131
Pinord	Probability packet arrives in order	135
Pdis	Probability packet is discarded	136

2. SIMPLE PROTOCOL WITHOUT ERROR CORRECTION

Perhaps the simplest communication system consists of a perfect (error free) line between two users with bandwidth B bits/sec and nearly constant propagation delay Tprop. With no errors, overhead for headers, or flow control, users simply transmit data over this line, obtaining a maximum throughput TPmax = B bits/sec. The mean delay to deliver a packet of length P bits is:

$$T = P/B + Tprop = \text{transmission time} + \text{propagation time} \qquad (1)$$

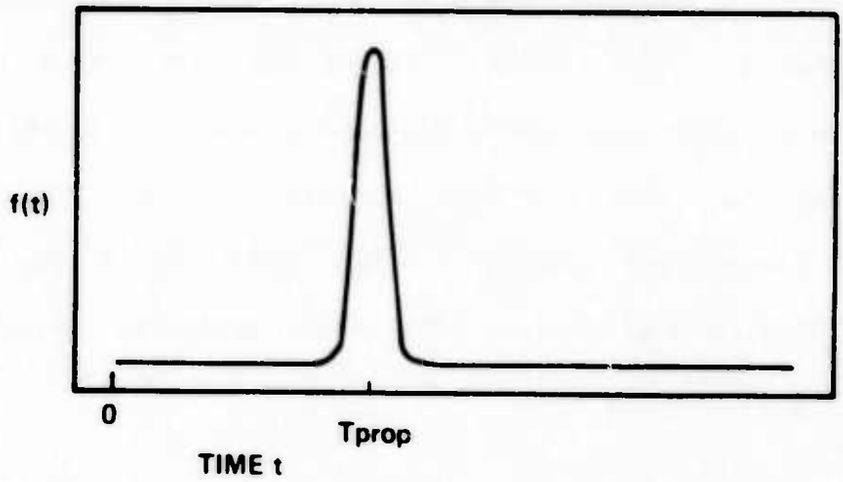
The line efficiency this ideal system is 1, giving a transmission cost of 1 bit/bit.

To increase the generality of this model, we will represent the propagation time for a packet as a probability density function, f(t). To represent a nearly constant delay Tprop, f(t) has a narrow, high peak at time t=Tprop (see figure 1a).

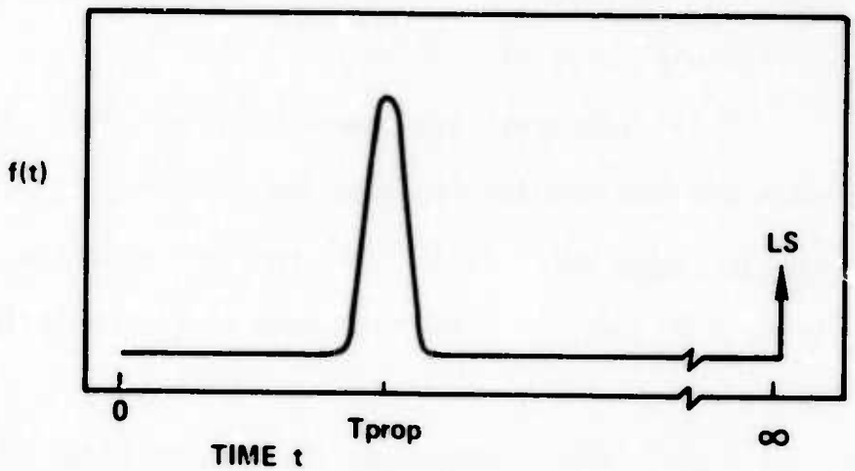
We also introduce the possibility of transmission errors, and assume that damaged packets are detected and discarded as described for PAR protocols in chapter II, but as yet no positive or negative acknowledgements (ACKs or NACKs) are returned for received packets. The probability, LS, of lost or damaged packets (which may depend on the packet length) can be included in f(t) as an impulse at t=infinity with value LS (the probability that a packet never arrives) (see figure 1b).

FIGURE 1 TRANSMISSION DELAY DENSITY FUNCTION f(t)

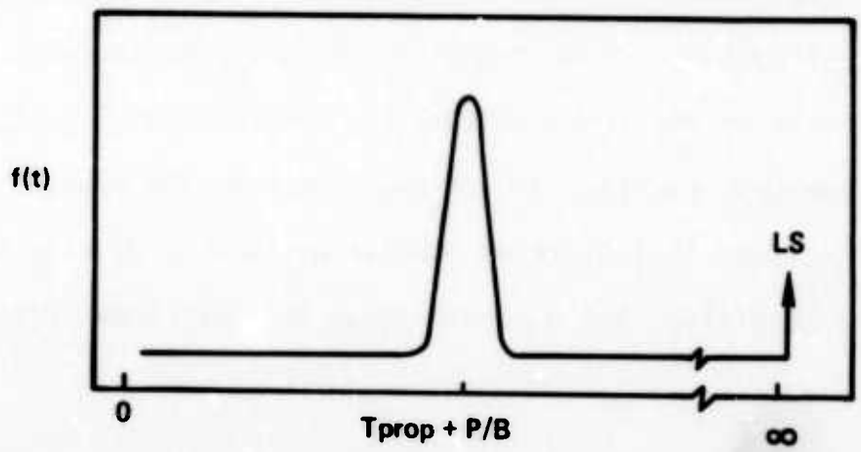
(a) PROPAGATION DELAY DENSITY FUNCTION f(t) WITH NO PACKET LOSS



(b) PROPAGATION DELAY DENSITY FUNCTION f(t) WITH PACKET LOSS PROBABILITY LS



(c) TRANSMISSION DELAY DENSITY FUNCTION f(t) INCLUDING PACKET TRANSMISSION TIME P/B AND PACKET LOSS PROBABILITY LS



Since we are primarily interested in PSN where the end-to-end propagation time is much larger than the Host to Packet Switch transmission time for a packet, it is convenient to also include the transmission time for a packet of length P in f(t) which will now also depend on P (see figure 1c). We call the end-to-end (or roundtrip) delay distribution including packet transmission time and packet loss or damage probability the transmission delay density function, f(t).

The source can still transmit packets at rate B/P, but only the fraction (1-LS) arrive successfully at the destination. Hence the average maximum throughput is given by:

$$TP_{max} = B \cdot (1-LS) \qquad (2)$$

The mean delay is strictly infinite since some packets (LS≠0) never arrive. The line efficiency is (1-LS), but the transmission cost is not well defined since some data is never delivered.

3. PAR PROTOCOL (RETRANSMISSION)

The most important reliability performance goal discussed in chapter II is to deliver each packet precisely once. In a PSN environment where packet loss and duplication occur, reliable communication requires a PAR type protocol as described in section II-3. Adding the constraint that every packet must be successfully delivered requires analysis of retransmission and control overhead necessary for reliable communication. This introduces the retransmission time interval parameter, R: if an ACK is not received within time R after a packet's last transmission, the packet will be retransmitted.

To provide error and duplicate detection with a PAR protocol, each packet must carry some control information, or a header of length H in addition to data. The header typically includes a checksum for error detection, and an identifier or sequence number for duplicate detection. It may also include address information, reverse ACKs, text length, or flow control information. In general the header length is fixed so P=H+D where D is the (variable) data length. The fraction of each packet taken up by the header will be called overhead, OH = H/P, which varies from H/Pmax of a few percent, to H/H=1 for control packets with no data. The throughput obtained due to other considerations must be multiplied by a factor TPoh to account for the portion of bandwidth consumed by overhead:

$$TPoh(P,H) = 1-H/P \qquad (3)$$

Retransmission introduces the first real difficulty in analyzing quantitative protocol performance. Now we must find the probability distribution for the first successful delivery of possibly many (re)transmissions. To do so, we assume that retransmissions take precedence over new transmissions, and hence when a packet's retransmission time arrives, it is immediately retransmitted. Preemption of a partially transmitted packet is not allowed, so a retransmission may actually have to wait for completion of a transmission in progress, but we assume this waiting time is insignificant compared to the retransmission interval R. This is a reasonable assumption in a PSN where R is typically an order of magnitude larger than a packet transmission time, P/B.

We also assume that the end-to-end delay density function f(t) and its associated cumulative distribution F(t) are identical for each (re)transmission of a packet, i.e. the delays for (re)transmissions of a packet are independent. This assumption is reasonable for large retransmission intervals typical of PSN's where alternate routing and long paths minimize dependence [Forgie75].

We can now write the successful transmission delay distributions including retransmission, g(t) and G(t), in terms of R and the basic transmission delay distributions f(t) and F(t) directly from basic probability considerations:

$G(t)$ = Prob(at least one successful delivery by time t)

= 1 - Prob(no success by time t)

= 1 - Prob(1st transmission not arrived
and 2nd trans. not arrived ...
... and nth trans. not arrived) $\quad n = \lceil t/R \rceil$

$$= 1 - \prod_{i=1}^{n} \text{Prob(ith trans. not yet arrived)}$$

$$= 1 - \prod_{i=0}^{n-1} [1-F(t-i \cdot R)] \qquad (4)$$

$g(t)$ = Prob(first successful delivery occurs at time t)

$$= \sum_{i=1}^{n} \text{Prob(trans. i arrives at time t and no other transmission yet arrived)} \quad n = \lceil t/R \rceil$$

$$= \sum_{i=0}^{n-1} \left(f(t-i \cdot R) \cdot \prod_{\substack{j=0 \\ j \neq i}}^{n-1} [1-F(t-j \cdot R)] \right) \qquad (5)$$

Of course $g(t) = d/dt\ G(t)$ as required for any probability distribution with the understanding that $G(t)$ is at least piecewise continuous (may abruptly change slope at points $t = i \cdot R$) so that $g(t)$ may have step discontinuities at points $t = i \cdot R$. Using equation 5 directly, it is also possible to determine the probability mass function $g(t)$ for a discrete distribution $f(t)$.

The mean delay including retransmission until the first successful delivery is given by:

$$DL(R,F) = \int_0^\infty t \cdot g(t) \cdot dt \qquad (6)$$

$$= \int_0^\infty [1-G(t)] \cdot dt$$

Another important measure is the <u>mean number of transmissions</u> until the first success:

$$Ntrans(R,F) = \sum_{i=1}^{\infty} i \cdot Prob(\underline{first} \text{ success between transmission } i \text{ and } i+1)$$

$$= \sum i \cdot (G(i \cdot R) - G((i-1) \cdot R))$$

which telescopes to

$$= \lim_{n\to\infty} [n \cdot G(n \cdot R) - \sum_{i=0}^{n-1} G(i \cdot R)]$$

Noting that $\lim_{n\to\infty} G(n \cdot R)$ must be 1 gives

$$= \lim_{n\to\infty} \sum_{i=0}^{n-1} [1 - G(i \cdot R)]$$

$$= \sum_{i=1}^{\infty} [1-G(i \cdot R)] \qquad (7)$$

If F(t) represents the roundtrip delay distribution (time from transmitting a packet until first ACK received), then equation 6 gives the mean roundtrip delay for a <u>successful</u> (acknowledged) transmission as a function of the retransmission interval R. Equation 7 gives the mean number of transmissions to achieve successful transmission as a function of R. Typically packet communication costs are dominated by a per packet charge, so Ntrans is also a good cost measure. Since

each successful transmission requires on the average Ntrans actual transmissions, maximum throughput attainable will be proportional to a retransmission factor:

$$TPretrans = 1/Ntrans \quad (8)$$

In general, a larger retransmission interval R allows higher throughput since no bandwidth is "wasted" retransmitting packets that might be long delayed and not actually lost. This is true because the source can continue sending new packets while it waits for ACKs of delayed packets (i.e. no sequencing of packets is performed).

Smaller R reduces mean delay for two reasons:

1) A Loss factor. Packets actually lost or damaged are retransmitted sooner.

2) An OR factor. Since all retransmissions are equivalent, the OR function in accepting retransmissions selects the minimum transmission time. The more retransmissions in progress at once, the smaller the minimum time for one.

The remainder of this section examines several representative transmission delay distributions, f(t), to explore the resulting protocol performance as a function of the retransmission interval R. The mean of each f(t) is fixed at unity to facilitate comparison, while shapes and variances of f(t) are varied. In each case, the resulting successful

transmission delay distributions, g(t) and G(t) from equations 5 and 4, are plotted for several values of R and packet loss probability, LS. Then equations 6 and 8 are used to plot delay and throughput as functions of R and LS. Finally, delay versus throughput is plotted for each f(t).

3.1 Constant Transmission Delay

Figures 2 and 3 show the successful transmission delay distributions g(t) and G(t) resulting from a constant transmission delay function F(t) with constant delay D=1 and loss probability LS:

$$F(t) = \begin{cases} 0 & t < D \\ 1-LS & D \le t < \infty \\ 1 & t=\infty \end{cases}$$

For this simple F(t), analytic results are easily derived for the mean delay until successful transmission, DL, and number of transmissions, Ntrans:

$$DL(D,R) = D + R \cdot LS/(1-LS)$$

$$Ntrans(D,R) = 1/(1-LS) + \lfloor D/R \rfloor$$

These results include the expected sum of a geometric series, 1/(1-LS), since in this case transmission is just a repeated series of independent trials, each with probability LS of failure. Mean delay DL is just the fixed delay D plus a term

FIGURE 2 SUCCESSFUL TRANSMISSION DELAY PROBABILITY MASS FUNCTION, g(t), FOR CONSTANT TRANSMISSION MEDIUM DELAY D = 1

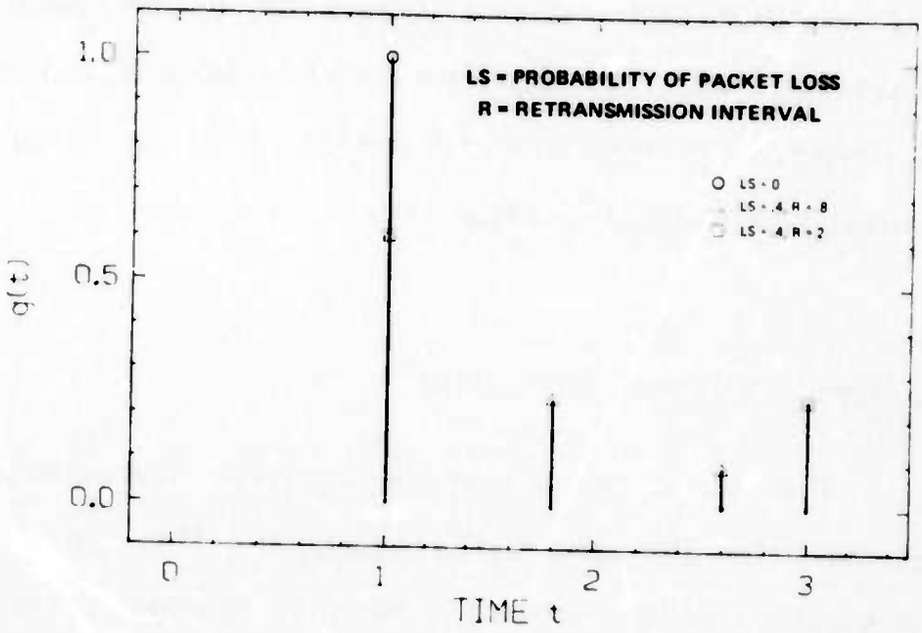
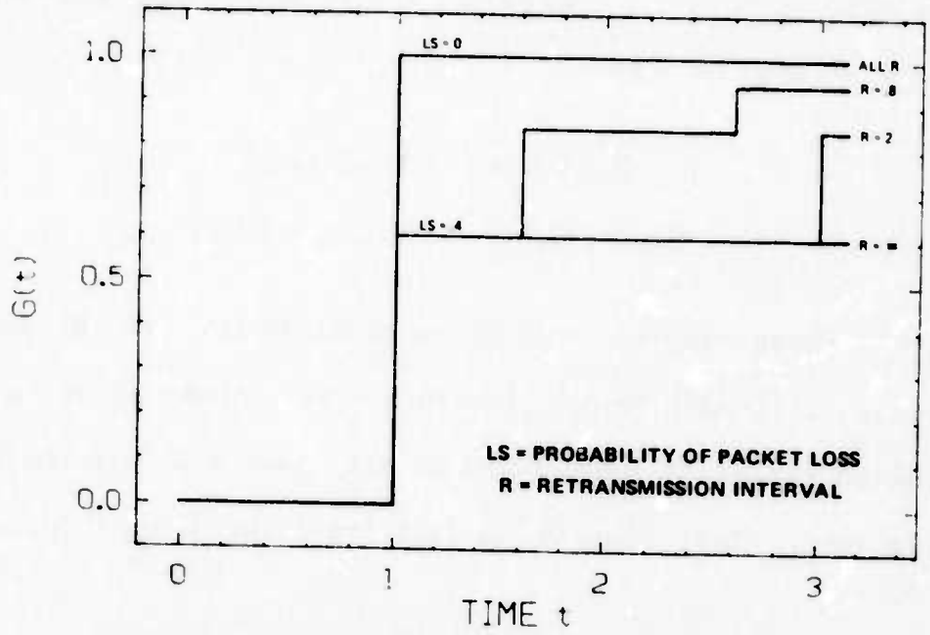


FIGURE 3 SUCCESSFUL TRANSMISSION DELAY CUMULATIVE DISTRIBUTION, G(t), FOR CONSTANT TRANSMISSION MEDIUM DELAY D = 1



proportional to the retransmission interval R. Only the loss factor operates to lower delay; since F(t) is constant, there is no overlap in f(t) from subsequent transmissions and the OR factor is zero. Figure 4 shows mean delay DL as a function of R for D=1. The delay for a constant F(t) gives the upper bound for delay resulting from other F(t) with nonzero variance where the OR factor does contribute to reducing DL.

Ntrans is just the mean number of trials for a Bernoulli process, 1/(1-LS), plus the additional number of trials executed until the success becomes "known" time D later. Figure 5 shows the mean throughput, TPretrans, as a function of R.

Figure 6 shows delay versus throughput resulting from a constant F(t) with D=1. For realistic error rates (LS<<1), R=D is clearly the optimal retransmission interval since there is no throughput gain by waiting longer than D, and little delay gain for retransmitting before time D. A constant transmission delay function presents an unrealistically narrow delay distribution, but it does capture the minimum delay behavior typical of PSN's.

3.2 Exponential Transmission Delay

Figures 7 and 8 show g(t) and G(t) resulting from an exponential transmission delay function (with mean delay = 1):

FIGURE 4

MEAN DELAY DL vs. RETRANSMISSION INTERVAL R FOR CONSTANT TRANSMISSION MEDIUM DELAY D = 1

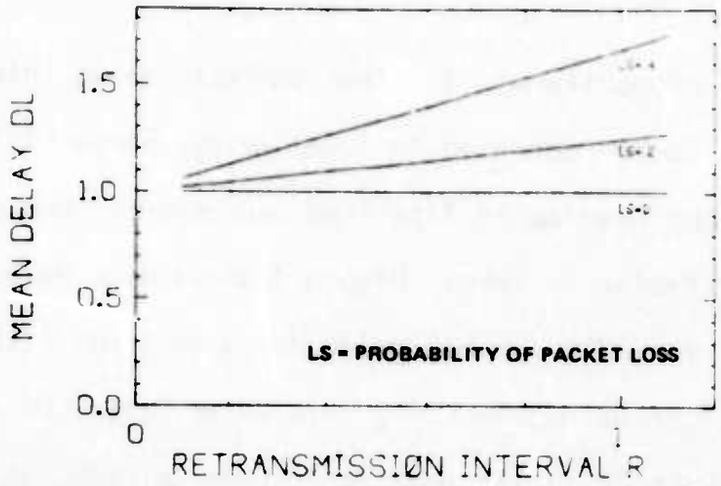


FIGURE 5

THROUGHPUT FACTOR TPretrans vs. RETRANSMISSION INTERVAL R FOR CONSTANT TRANSMISSION MEDIUM DELAY D = 1

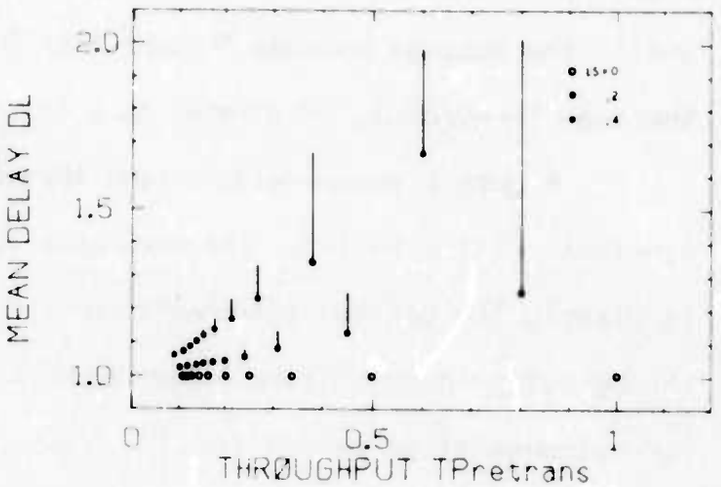


FIGURE 6

MEAN DELAY DL vs. THROUGHPUT FACTOR TPretrans FOR CONSTANT TRANSMISSION MEDIUM DELAY D = 1

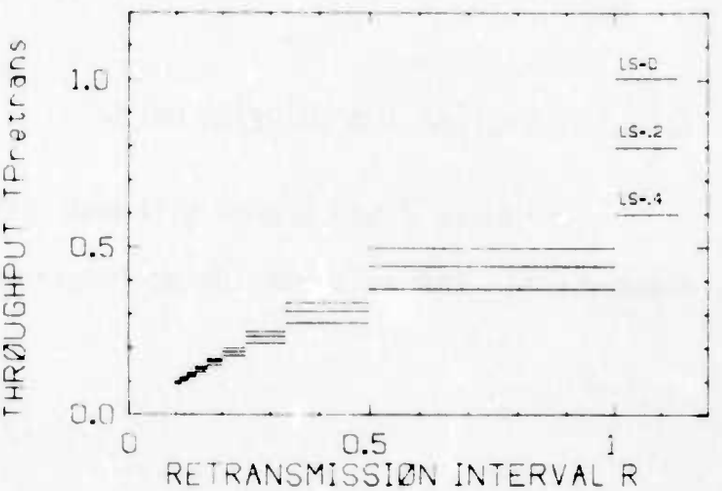
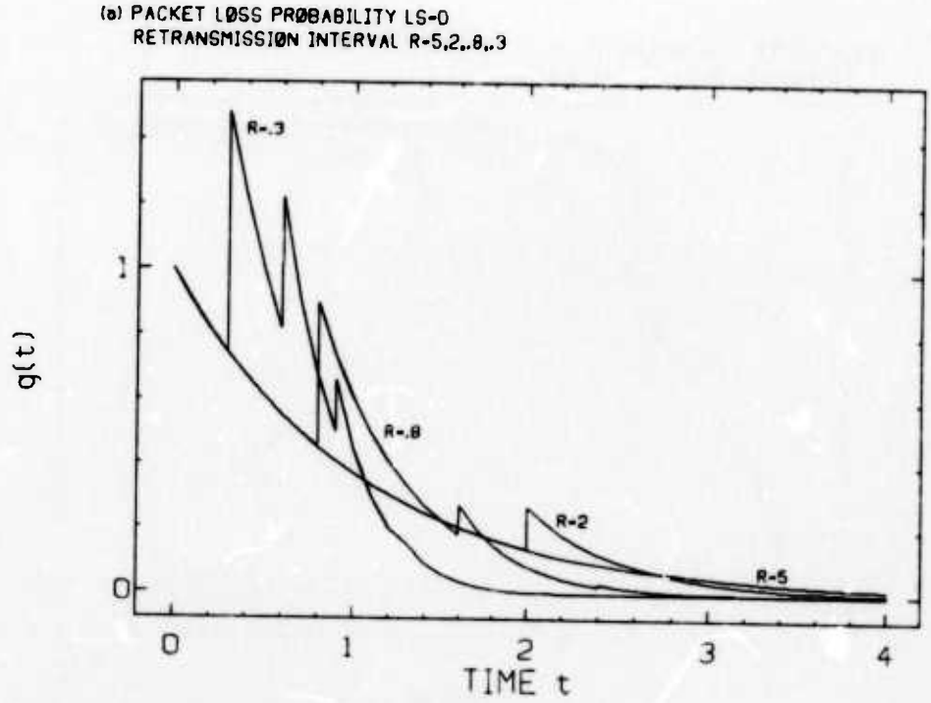


FIGURE 7 SUCCESSFUL TRANSMISSION DELAY PROBABILITY DENSITY FUNCTION, g(t), FOR EXPONENTIAL TRANSMISSION MEDIUM DELAY WITH MEAN = 1



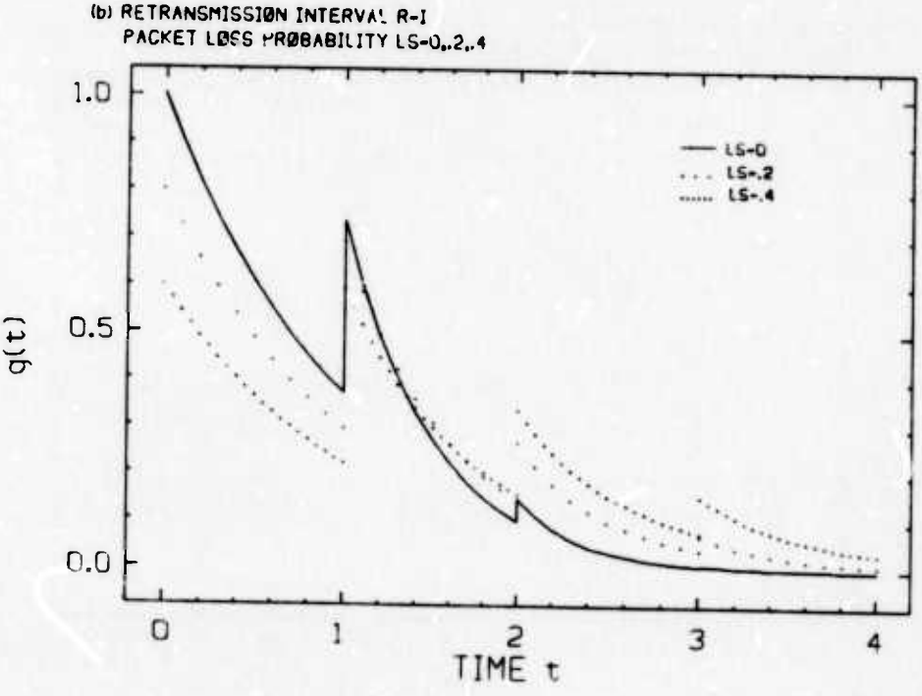
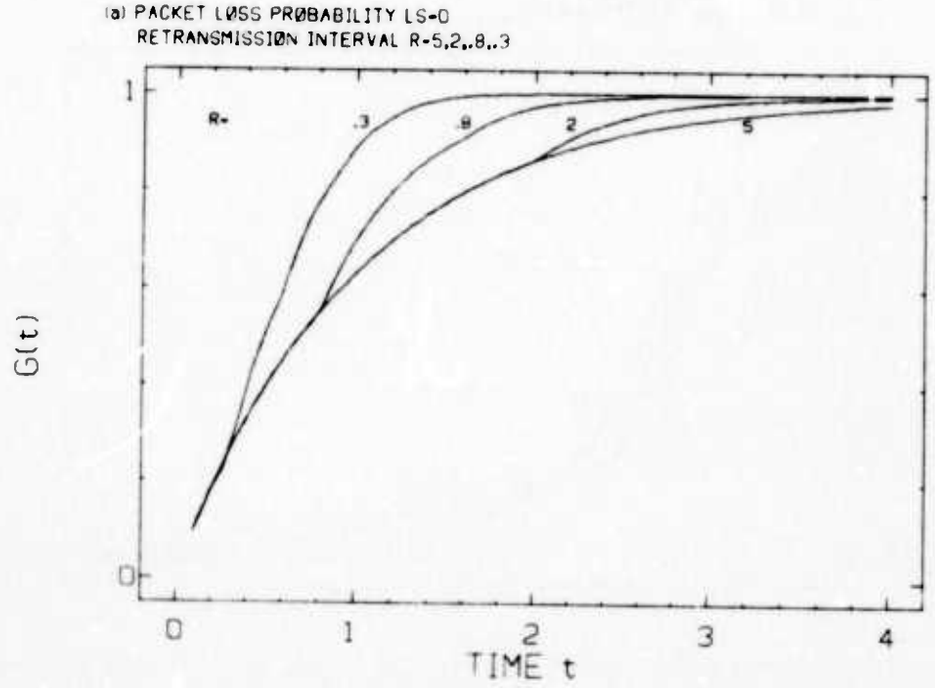
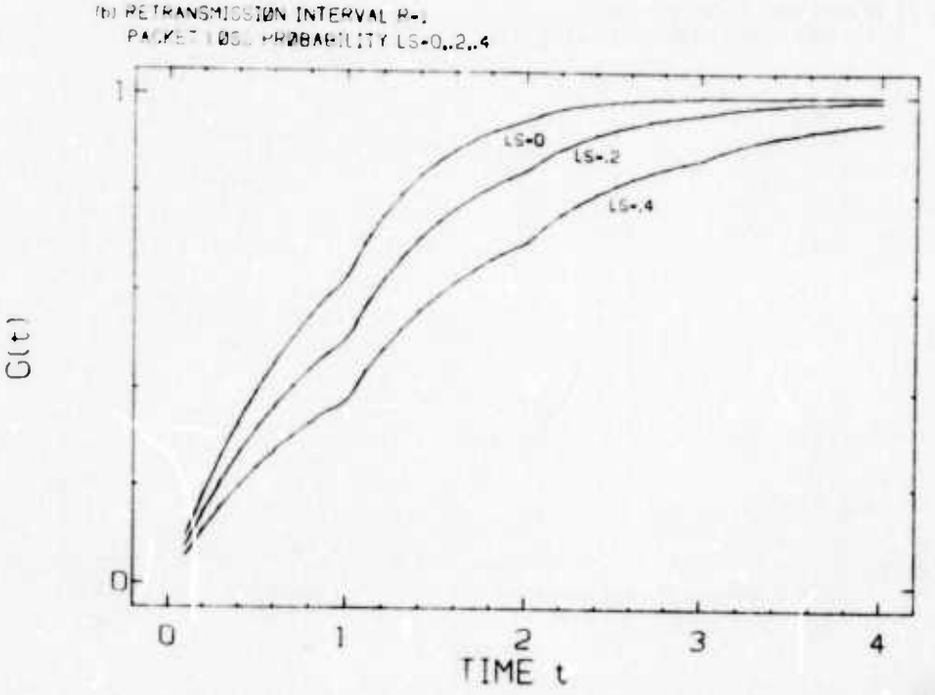


FIGURE 8 SUCCESSFUL TRANSMISSION DELAY CUMULATIVE DISTRIBUTION, G(t), FOR EXPONENTIAL TRANSMISSION MEDIUM DELAY WITH MEAN = 1





$$F(t) = \begin{cases} (1-LS)(1-e^{-u \cdot t}) & 0 \le t < \infty \\ 1 & t = \infty \end{cases}$$

Figures 7a and 8a show the successful transmission delay distributions, g(t) and G(t), for several retransmission intervals R but no packet loss (LS=0). Smaller R moves the delay density toward shorter times because of the significant OR factor with the wide exponential f(t). Figures 7b and 8b show g(t) and G(t) for several packet loss probabilities LS at a fixed R. Smaller LS also moves the delay density to the left.

For LS=0, equations 6 and 7 readily yield analytic expressions for mean delay DL and number of transmissions Ntrans:

$$DL(u,R) = \frac{1}{u} \cdot \left(1 - \sum_{i=1}^{\infty} \left[\frac{1}{i \cdot (i+1)} \cdot e^{-i \cdot (i+1) \cdot R \cdot u/2}\right]\right)$$

$$Ntrans(u,R) = \sum_{i=0}^{\infty} e^{-i \cdot (i+1) \cdot R \cdot u/2}$$

For nonzero LS, numerical solution techniques become necessary.

Figures 9 and 10 show mean delay DL and throughput TPretrans for various R and LS. Results from the previous section are shown dotted for comparison. The OR factor serves to lower delay for small R because the wide exponential f(t) for neighboring transmissions overlap significantly for small R. As R increases, mean delay approaches the upper bound proportional

FIGURE 9

MEAN DELAY DL vs. RETRANSMISSION INTERVAL R FOR EXPONENTIAL TRANSMISSION MEDIUM DELAY WITH MEAN = 1

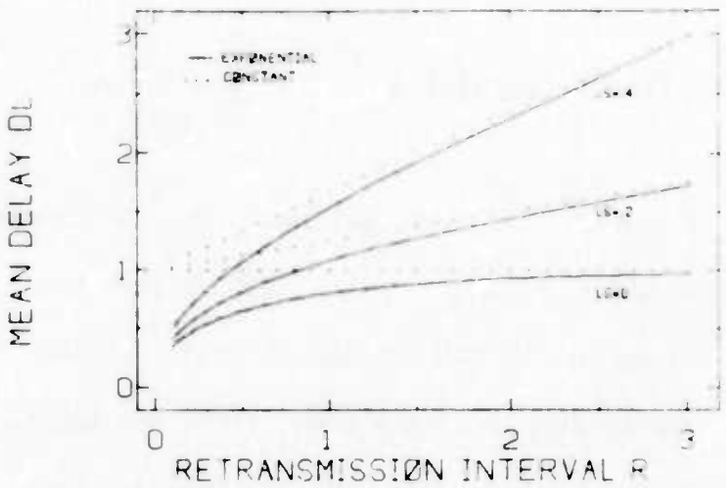


FIGURE 10

THROUGHPUT FACTOR TPretrans vs. RETRANSMISSION INTERVAL R FOR EXPONENTIAL TRANSMISSION MEDIUM DELAY WITH MEAN = 1

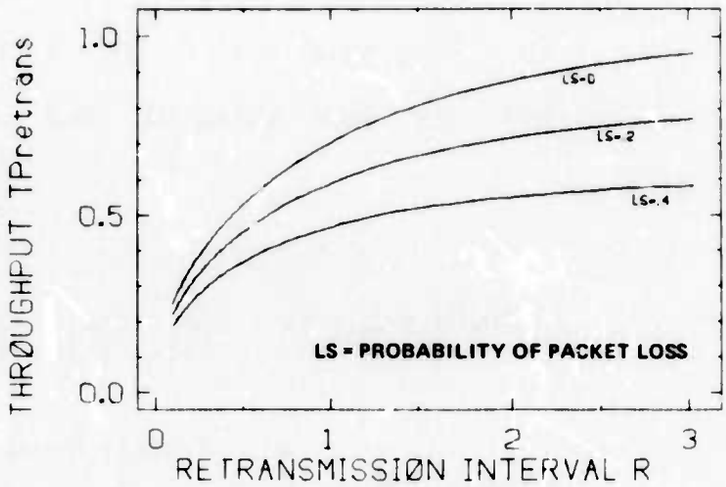
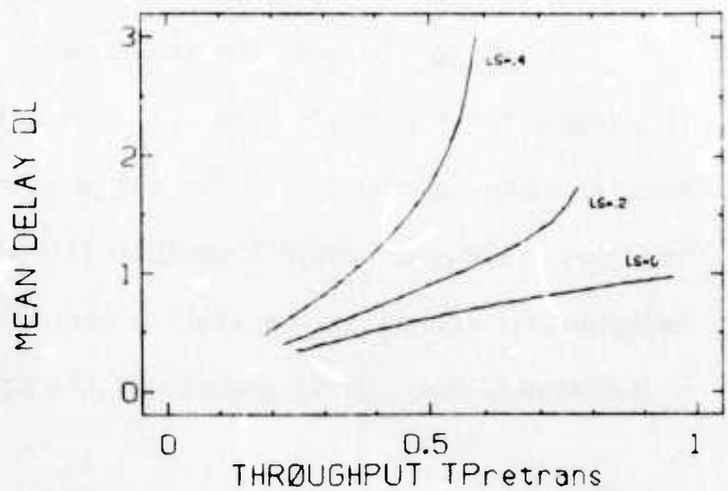


FIGURE 11

MEAN DELAY DL vs. THROUGHPUT FACTOR TPretrans FOR EXPONENTIAL TRANSMISSION MEDIUM DELAY WITH MEAN = 1



to R as with a constant f(t) where only the Loss factor is contributing. Throughput rises smoothly with R because of the wide spread of f(t).

The exponential transmission delay function presents a wide delay distribution but does not incorporate a minimum delay, opposite to a constant f(t). There is no optimal operating point on the throughput vs. delay curves shown in figure 11, but rather a smooth tradeoff of throughput for delay.

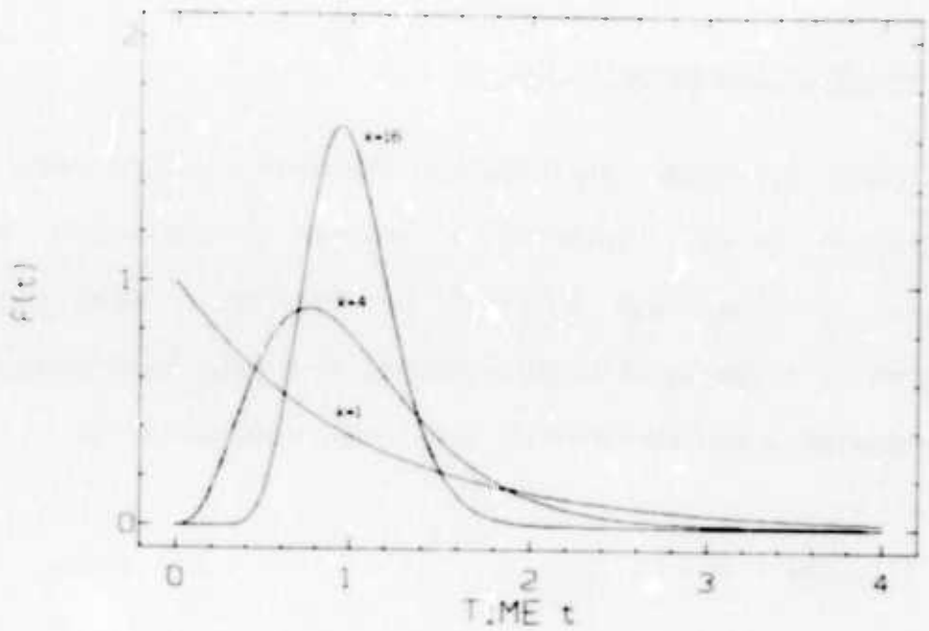
3.3 Erlangian Transmission Delay

The Erlangian distribution represents a more realistic transmission delay, including a minimum transmission time, moderate variance, and a small but long tail. Actually the Erlangian is a family of distributions, with mean determined by the parameter u and variance by the "shape" parameter k:

$$f(t) = \begin{cases} (1-LS) \cdot (k \cdot u) \cdot \dfrac{(k \cdot u \cdot t)^{k-1}}{(k-1)!} \cdot e^{-k \cdot u \cdot t} & 0 \le t < \infty \\ LS \cdot (\text{unit impulse at } t=\infty) \end{cases}$$

The mean of the Erlangian distribution is 1/u while the variance with mean of unity is just 1/k. This family conveniently models a wide range of delay distributions from exponential (k=1) to constant (k→∞). Figure 12 shows the Erlangian f(t) with mean=1 and k=1,4,16.

FIGURE 12 ERLANGIAN PROBABILITY DENSITY FUNCTION, f(t), WITH MEAN = 1 AND SHAPE PARAMETER k = 1, 4, 16

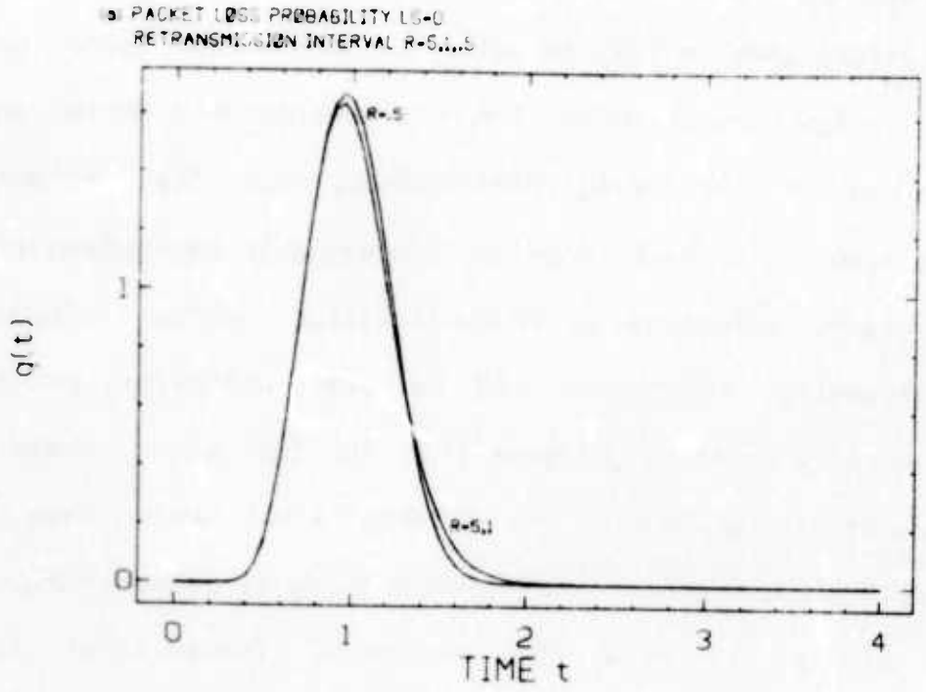


Several authors have measured ARPANET mean delay times under various conditions [Kleinrock74a, Cole71, Naylor73] and more recently Forgie (1975) has obtained transmission delay distributions under a limited set of circumstances. Even under these limited circumstances, there is considerable variation in the spread of the delay distribution, but the Erlangian distribution with k=16 provides a reasonable approximation to real network transmission characteristics while remaining computationally manageable. As we shall see below, protocol performance is relatively insensitive to the exact shape or variance of f(t) as long as the variance is not larger than one, so a perfect representation of network delay is unnecessary.

Figure 13 shows the successful transmission delay distribution g(t) resulting from an Erlangian f(t) with mean=1 and k=16. Figure 14 shows the cumulative delay distribution G(t) for several retransmission intervals R and loss probabilities LS. Again smaller R and LS move the distribution to the left (shorter times) although not as much as with exponential f(t).

Figure 15 shows mean delay DL as a function of R for several LS. Results from the previous section are shown dotted for comparison. The Loss factor and the OR factor both serve to reduce delay for small R, but the OR factor is much less pronounced than for exponential f(t) since the delay density is more concentrated about the mean. For large R, mean delay again

FIGURE 13 SUCCESSFUL TRANSMISSION DELAY PROBABILITY DENSITY FUNCTION, g(t), FOR ERLANGIAN TRANSMISSION MEDIUM DELAY WITH MEAN = 1 AND k = 16



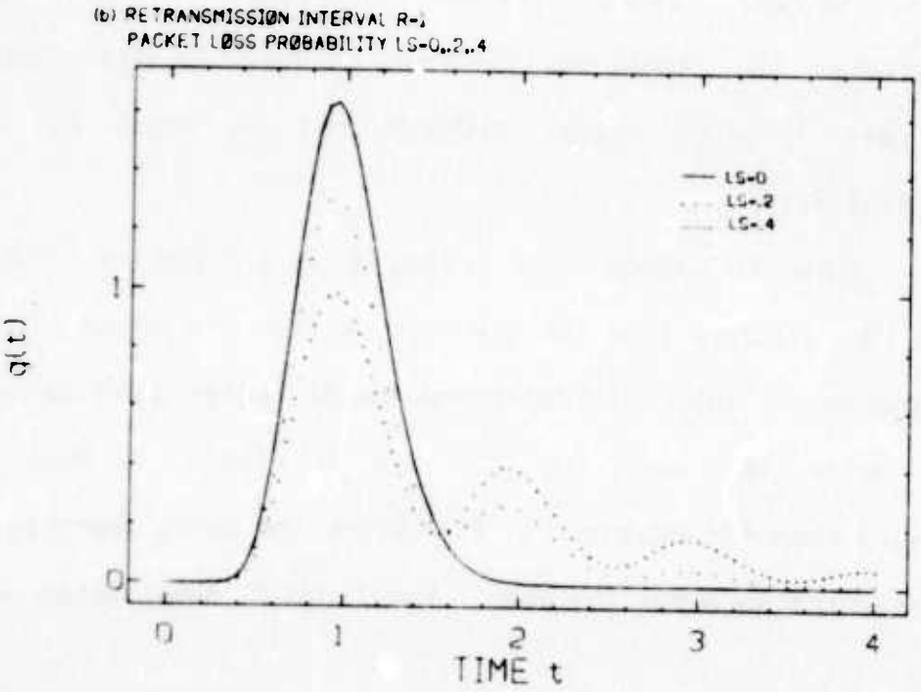
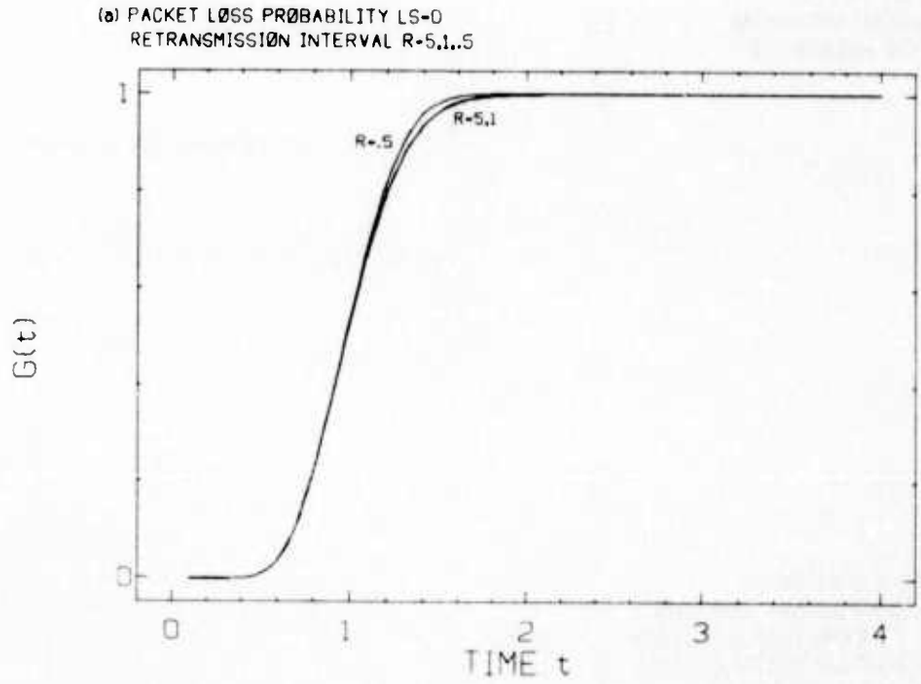


FIGURE 14 SUCCESSFUL TRANSMISSION DELAY CUMULATIVE DISTRIBUTION, G(t), FOR ERLANGIAN TRANSMISSION MEDIUM DELAY WITH MEAN = 1 AND k = 16



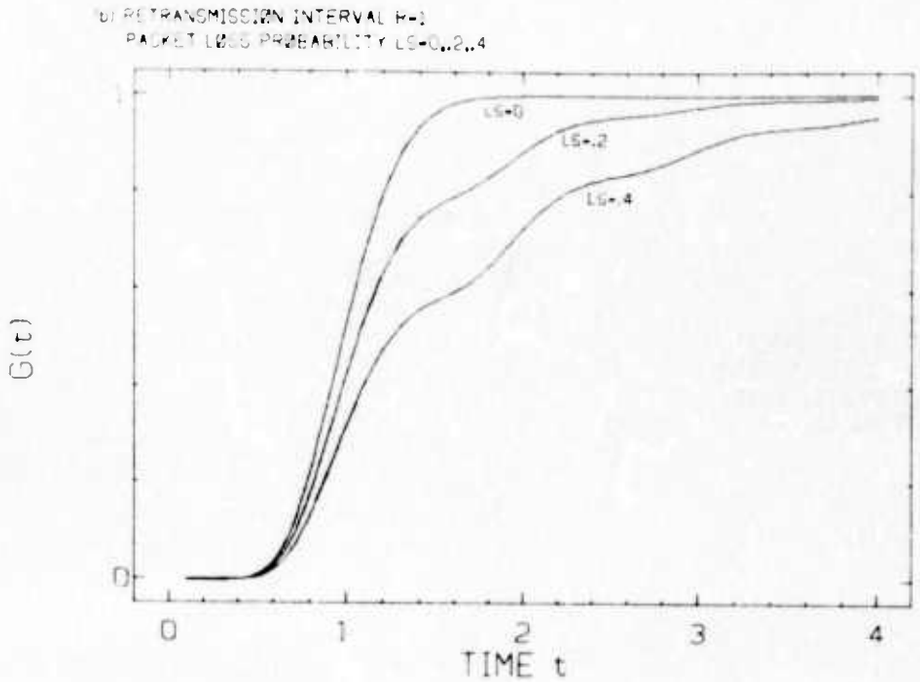


FIGURE 15

MEAN DELAY DL vs. RETRANSMISSION INTERVAL R FOR ERLANGIAN TRANSMISSION MEDIUM DELAY WITH MEAN = 1 AND k = 16

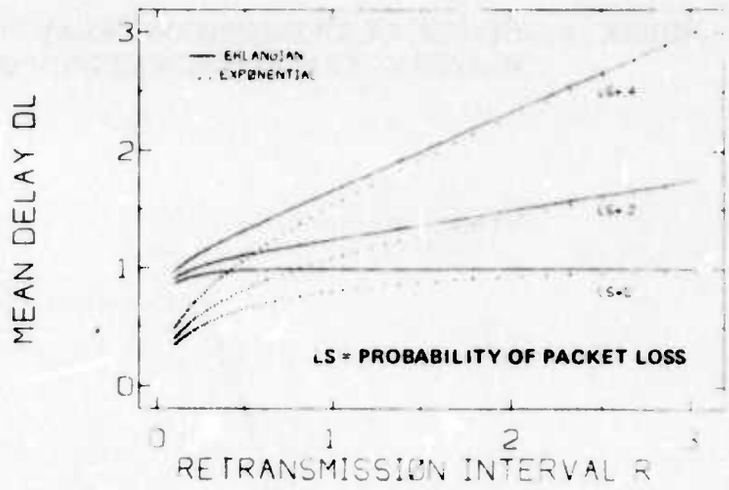


FIGURE 16

THROUGHPUT FACTOR TPretrans vs. RETRANSMISSION INTERVAL R FOR ERLANGIAN TRANSMISSION MEDIUM DELAY WITH MEAN = 1 AND k = 16

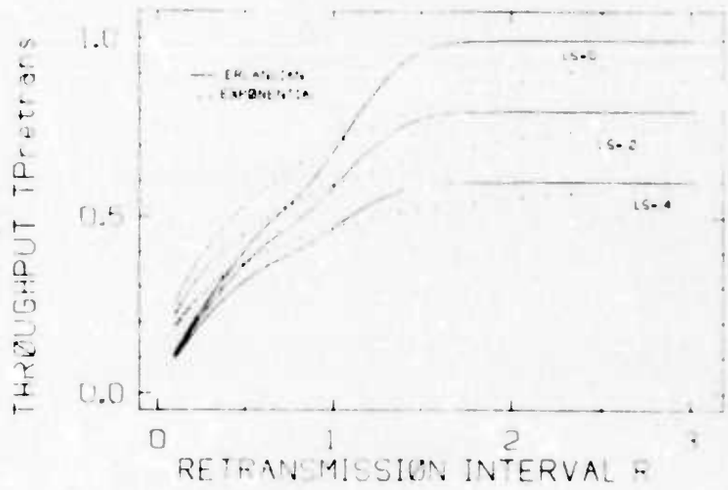
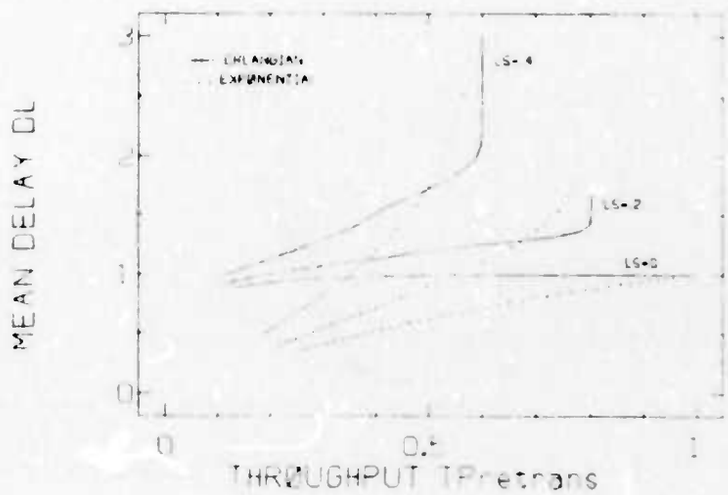


FIGURE 17

MEAN DELAY DL vs. THROUGHPUT FACTOR TPretrans FOR ERLANGIAN TRANSMISSION MEDIUM DELAY WITH MEAN = 1 AND k = 16



approaches the upper bound proportional to R as for a constant f(t) where the OR factor does not contribute at all.

Figure 16 shows mean throughput TPretrans as a function of R for several LS. Throughput resulting from the moderate variance Erlangian f(t) with k=16 is already approaching the step-like behavior derived for a constant f(t), with faster approach to the limiting throughput for R>1 than with the wider exponential f(t).

Finally, figure 17 shows delay versus throughput resulting from the Erlangian f(t) with mean=1 and k=16. For nonzero packet loss probabilities, a definite "knee" occurs because delay increases linearly with R while throughput quickly approaches its maximum with increasing R.

3.4 Results

We have examined PAR protocol performance resulting from varying the retransmission interval R with a wide range of transmission delay distributions f(t) and packet loss probabilities LS. Mean delay DL rises linearly with R and LS for realistic values as expected in a "repeat until success" system. For R<1, DL drops somewhat more quickly due to the OR factor described above. However, this effect is only significant with high variance f(t), and is accompanied by a large increase in the average number of transmissions required, and hence a decrease in attainable throughput.

A throughput factor TPretrans equal to the inverse of the number of transmissions required was defined and represents the maximum average throughput attainable with a given R, taking into account the fraction of bandwidth used in retransmission. TPretrans asymptotically approaches its maximum of 1/(1-LS) for large R. With realistic f(t) this results in the "knee" or optimal performance area observed in delay versus throughput curves. The location of this knee is determined primarily by the mean and variance of f(t), and not by loss probability LS. The knee is sharpest for small variances and occurs at a value of R such that R is also the knee of the F(t) curve (i.e. the packet has almost certainly arrived if it is going to arrive, by time R after transmission).

In summary, the best strategy for choosing a retransmission interval R is to set R equal to the time when "most" transmissions would have succeeded if there were no lost or damaged packets. Larger R brings minimal improvement in attainable throughput while increasing delay. Smaller R brings significant throughput degradation with minimal decrease in delay. However, for low total throughput requirements, mean delay may be reduced by using a smaller R, but with a substantial cost in additional retransmission.

For realistic error rates (LS<<1), mean delay is quite insensitive to R, so a relatively wide range of R is near optimal. Since network transmission delay varies with time,

using a somewhat larger fixed R is probably a good heuristic to stay in the high throughput portion of the performance curve. R may also be set dynamically on the basis of observed transmission delays.

We are now able to include the effects of retransmission and overhead in protocol performance. Equation 6 gives the delay resulting from choice of R. The maximum average throughput attainable, TPmax, is a product of the overhead factor TPoh from equation 3, the retransmission factor TPretrans from equation 8, and the transmission medium bandwidth B:

$$TPmax = TPoh \cdot TPretrans \cdot B \qquad (9)$$

4. FLOW CONTROL

In section 3 we found the throughput limitation due to retransmission of packets by deriving the fraction of available bandwidth consumed by retransmissions. Another throughput limitation results when roundtrip delay is large relative to packet transmission time as is frequently the case in packet switching networks. In this case, the sender may be idle a large fraction of the time waiting for an acknowledgement.

To achieve higher throughput, the sender may be allowed to transmit multiple packets before receiving any acknowledgements. Since each outstanding packet requires buffer storage and other source resources, an importatant efficiency question becomes how large must the "window" of allowed transmissions be in order to achieve maximum throughput?

For several reasons it is also desirable and even imperative for source transmission rate to be limited. The transmission medium itself may become congested due to excessive traffic from all Hosts it serves, requiring some means of congestion control to limit entering traffic. Several techniques have been proposed to deal with network congestion [Kahn72, McQuillan72, Davies72, Pouzin73b, Belsnes74, Crowther75]. These constraints are generally enforced by the transmission medium and are not under control of a communication protocol so we do not discuss them further.

More relevant to this study, transmission rate between each source and destination process must be controlled to match a sender's production rate to the receiver's consumption rate, minimizing buffer storage and bandwidth requirements for the resulting throughput. This presents the main problem of interprocess flow control and suggests a second important efficiency question: How small should the window of allowed transmissions be in order to limit source transmission to a given rate? Several authors have discussed techniques for end to end flow control in PSN [Kahn72, Walden72, Carr70, Zimmerman75, Pouzin74c, Cerf74c, Cerf75, Belsnes74, Crowther75, Opderbeck74], but quantitative results have been lacking. In many cases, results are complicated by sequencing or reassembly requirements discussed in section 6.

Most strategies can be described in terms of a limited window size, Nwin, of allowed transmissions [Pouzin74c, Cerf74c]. In general a limit of Nwin packets (and/or bits) is imposed such that up to Nwin packets (bits) may be transmitted but not yet acknowledged at any moment. When the limit is reached, the sending discipline stops transmitting new packets until an ACK arrives, freeing space for new transmission. If R is exceeded for some pending packet, the packet may still be retransmitted.

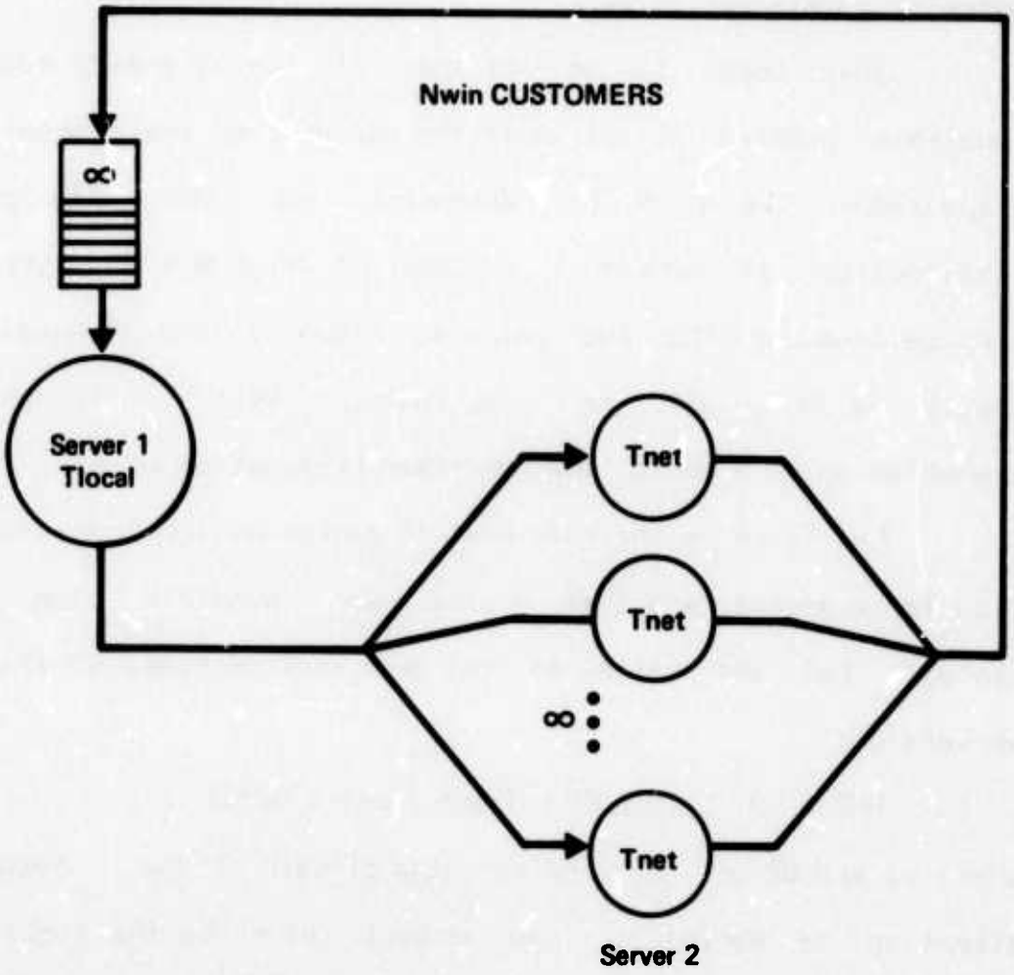
Strictly speaking, an arriving ACK functions only to signal the error recovery mechanism that a packet has

successfully been received and no further retransmissions are required. The credit granted for new transmission may be less than, equal to, or greater than the amount acknowledged. The separation of credits from ACKs is more fully discussed in the next section, while in this section we consider only the fixed window size case where an ACK implicitly grants permission to send a new packet.

The limit of Nwin pending packets (or bits) functions as flow control on the source of packets. We want to determine the relation of Nwin to achievable throughput so the receiver can select Nwin to limit throughput or achieve maximum throughput as desired. As a crude means of flow control, the receiver could simply discard arriving packets in excess of the rate desired, but this strategy wastes transmission medium capacity, degrading performance for other connections, and increases costs by increasing the retransmission required. Hence it is desirable to select Nwin to limit the sender's transmission rate so that essentially all packets arriving at the receiver can be accepted.

Figure 18 shows a closed network of queues with two servers that adequately represents the constant window size flow control model. Server 1 represents a source of packets to be transmitted serially into the net. The infinite server system 2 represents the (parallel) transmission of packets, processing at the destination, and return of ACKs through the net to server 1.

FIGURE 18 QUEUING MODEL OF FLOW CONTROL



There are Nwin "customers" in the system, so whenever all Nwin packets are in service at server 2, server 1 is idle. These idle periods represent the time a source would be blocked from transmitting new packets by the flow control mechanism. (Note that retransmissions of pending packets may occur during blocked times in the real protocol which are not represented in the queueing model--see below.)

This model is an instance of the classic machine repairman problem [Cox61] with the emphasis on the transmitter (repairman). We wish to determine how the throughput (utilization) of server 1 depends on Nwin and the ratio of service times of the two servers. Ideally the transmitter should be busy all the time, but if Nwin is too small, transmission is blocked (the repairman runs out of work).

Let Tlocal = the mean Host to packet switch transmission time for a packet, and Tnet = the mean roundtrip time less Tlocal. Let the ratio of the mean service times of the two servers be:

RHO = (ST 1)/(ST 2) = Tlocal/Tnet = u2/u1

where u1 and u2 are the service rates of each server. Focusing attention on server 1 (the sender), let n1 be the number of customers queued or in service at server 1, and Pi = Prob(n1=i). P0 is the probability that server 1 is idle (blocked from transmitting by the flow control mechanism). When there are i customers at server 1, there will be Nwin-i customers at server

2, and the arrival rate of new customers at server 1 will be u2·(Nwin-i). This shows that the closed system in figure 18 is equivalent to an open single server system with finite customer population Nwin. Kleinrock (1975) gives the expression for 1/P0 in this system:

$$1/P0 = \sum_{i=0}^{Nwin} \frac{Nwin!}{(Nwin-i)!} \cdot RHO^{i}$$

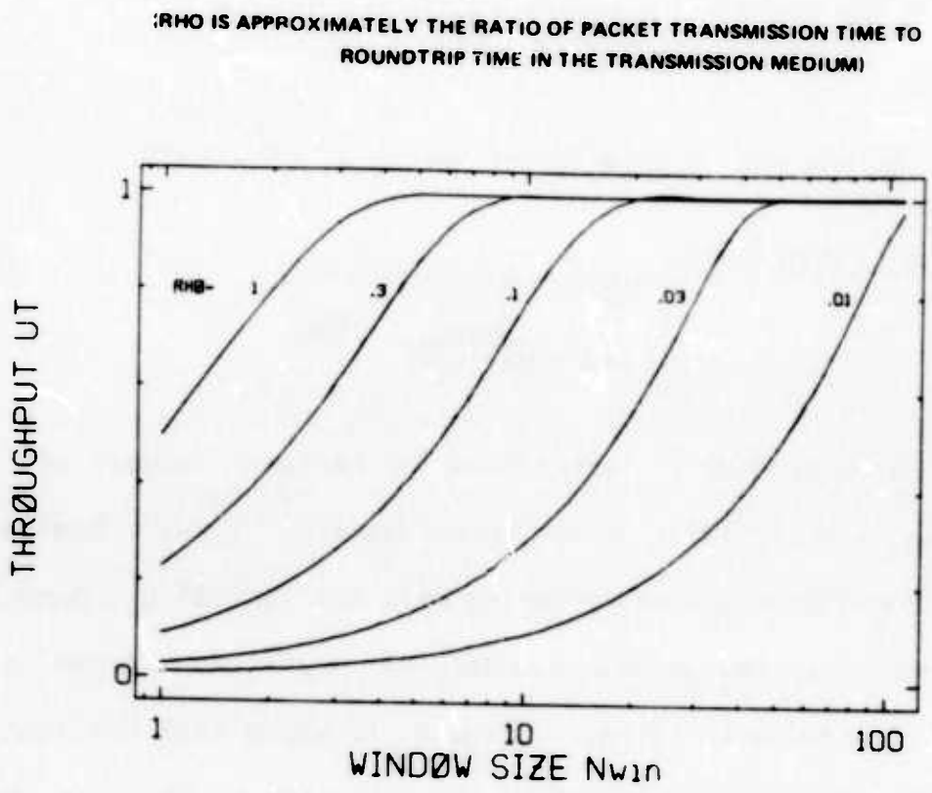
Finally define the utilization of server 1, UT = 1-P0:

$$UT(Nwin,RHO) = 1 - \frac{1}{\sum_{i=0}^{Nwin} \frac{Nwin!}{(Nwin-i)!} \cdot RHO^{i}} \qquad (10)$$

In figure 19 we plot UT versus Nwin for various values of RHO assuming exponentially distributed service times. Realistic values for RHO in packet switching nets are typically around 0.1 since roundtrip delays are an order of magnitude larger than packet transmission times. Figure 19 shows that utilization (throughput) rises approximately linearly with window size up to the half-way point of UT=0.5, and then more slowly approaches unity with increasing window size. For smaller RHO, a larger window size is necessary to keep the sender busy.

An upper bound on UT can be found by considering the deterministic system with constant service times: ST2=n·ST1 (Note n=1/RHO). In this system a window size of exactly

FIGURE 19 THROUGHPUT FACTOR UT vs. FLOW CONTROL WINDOW SIZE Nwin FOR VARIOUS RHO = Tlocal/Tnet



Nwin=n+1 is required to keep the sender busy (ignoring errors) since then the first ACK will be returning just as the n+1st packet is transmitted. For Nwin < n+1, UT=Nwin/(n+1). For other service distributions with coefficients of variation Cs between 0 (constant) and 1 (exponential), Cox and Smith (1961) suggest that the value of UT may be found by linear interpolation with Cs squared between constant and exponential values of UT:

$$UT = UTconst + Cs^2 \cdot (UTconst - UTexp)$$

We are now in a position to answer the two questions posed above:

1) How large a window, Nwinmax, is required to achieve maximum throughput?

Nwinmax = n+1 = approximately Tnet/Tlocal

This corresponds to the intuitive approach of "keeping the pipe full" between sender and receiver to achieve maximum throughput. For roundtrip delay distributions with larger variance, a slightly larger window is necessary.

2) How small a window should be used to limit throughput to a particular value? Throughput rises approximately linearly with window size up to Nwinmax. Thus to limit the sender to 0.1 nominal bandwidth B, the receiver should select a window size of approximately 0.1·Nwinmax.

As remarked above, the queuing model of flow control does not explicitly consider errors, retransmission, or overhead. Each packet transmitted carries an overhead OH (cf section 3). Furthermore, each customer served by server 1 actually consists of the original transmission followed by Ntrans-1 retransmissions. These retransmissions are not included in the utilization above, but must be included in determining the limiting performance. Assuming retransmissions at their assigned interval R take precedence over new transmissions, the total traffic generated equals UT·Ntrans which cannot exceed unity. Hence a smaller window size Nwin = Nwinmax/Ntrans will generate the maximum allowable traffic.

When UT, the rate of new data transmission, is small because of window size limitations , the "extra" bandwidth is available for any retransmissions necessary and throughput is flow control limited. When UT approaches one due to a large window size, retransmissions take precedence over new transmissions that might otherwise be allowed, and throughput is retransmission limited. Achievable throughput with both retransmission and flow control effects is the minimum allowed by either effect. Combining these results with equation 9, maximum average throughput attainable becomes:

$$TP_{max} = TP_{oh} \cdot \min(UT, TP_{retrans}) \cdot B \qquad (11)$$

Another way to interpret the limit Nwin is as a source imposed resource limitation. Since each packet transmitted must be stored at the source until an ACK returns, buffer space may become scarce when many connections become active. In sharing its processing and storage capacity among many connections, a protocol may limit the portion devoted to each connection and reduce the window size per connection below what might be allowed by the receiver. For example, an ARPANET TIP [Ornstein72] limits each normal terminal user to a window of 6-12 characters in order to share its relatively scarce buffer space among all users.

An alternative interpretation of the queuing model is to consider the number of type 2 servers limited to N, but unlimited customers, rather than limiting the total number of customers in the system and having infinite type 2 servers. This interpretation models the situation where network capacity is the blocking factor--once the Host has transmitted N packets, the network blocks further transmission until new permission to send is returned. This was precisely the situation in a recent ARPANET congestion control strategy: Source and destination IMPs imposed a fixed window size of four messages [McQuillan72] for traffic between each pair of Hosts.

Flow control constraints can affect total transmission delay as well as throughput since a transmission request may have to wait until window space is available. However, this waiting time is the type of throughput dependent delay mentioned in the introduction to this chapter and will not be considered further. Cochi (1973) has treated increased waiting time in systems of queues under similar circumstances where a server is blocked from further processing because the next service facility is full.

5. DESTINATION BUFFER ALLOCATION

The previous section considered the impact of source buffering constraints on protocol performance. In this and the following sections, we consider destination buffering requirements and the performance degradation resulting from limited destination buffer space. Many authors have discussed storage allocation for related communication problems [Chu74] such as switching node buffer requirements [McQuillan74, Closs73, Fultz72, Danthine75c] and terminal data buffering [Gaver71, Metcalfe73]. These analyses often assume constant transmission delays over simple transmission lines. The following analysis focuses on destination buffering strategies for end-to-end protocols in a PSN with highly variable transmission characteristics as discussed in chapter I.

Destination buffer or storage allocation policy is closely connected with flow control. In particular, it is often assumed that the window size and the buffer space allocated for receiving packets must be identical. In fact, under ideal circumstances when packets arrive uniformly spaced and in order, double buffering is adequate to handle an arbitrarily large window size and accompanying large throughput. Under these conditions, the process can consume and return one buffer while the other is being filled with an arriving packet. Real situations probably require something between the extremes of minimal and full buffering.

Larger storage allocation at the destination typically becomes necessary for two purposes:

1) Smoothing uneven production and consumption rates. This frequently occurs in multiprogramming systems where process activity occurs in bursts.

2) Reordering packets arriving out of order. A sequencing protocol must deliver packets in sequence, so early arrivals must be held until their predecessors arrive. Any fragments created between source and destination must also be reassembled before delivery. This is discussed in the next section.

Inadequate buffer space for either purpose results in correctly received packets being discarded because there is no place to put them. Throwing this effect into the loss factor LS in the delay distribution F(t) confuses all the earlier results which depend on F(t). It is more illuminating to preserve the assumption of guaranteed acceptance at the destination for previous results, and introduce an independent throughput degradation factor, TPbuf, for destination buffer allocation effects.

5.1 Acknowledgement and Buffer Allocation Strategies

As observed in the last section, return of an Acknowledgement to indicate successful receipt of a packet and suppress retransmission need not be tied to return of credits or permission to advance the window allowing new transmissions. In the simple implementations treated in section 4, Acknowledgements and credits must be returned together. Conservative systems typically delay returning credits until new receive buffer space has actually been made available by "consuming" arrived packets or furnishing new space [Zimmerman73]. This policy increases roundtrip delay (which includes destination processing time) and hence may reduce throughput for a given window size as shown in section 4. Of course this throughput limitation may be precisely what the receiver desires in matching the source's production rate to his own consumption rate.

Unfortunately, increasing roundtrip delay for Acknowledgements also results in more retransmissions, higher cost, and less efficient line use for a given retransmission interval R (cf section 3). R could be increased to compensative, but this increases delay when packets are lost. An alternative strategy involves returning Acknowledgements immediately on successful receipt of a packet, and flow control information at a possibly later time. In this case, the shorter

roundtrip delay for Acknowledgements is used in calculating an appropriate R, while the longer roundtrip time for credits governs throughput due to flow control. Overhead may increase when control information is returned separately for error recovery (Acknowledgements) and flow control (credits) [Kleinrock74].

This immediate acknowledgement strategy may reduce retransmissions, freeing transmission medium capacity for other users, but it does not alter throughput between its own users which is limited by the roundtrip time for credits. Acknowledgements also have a somewhat different meaning more like "received" than "processed" in this scheme.

Both of the above conservative strategies guarantee that an arriving packet will never have to be discarded for lack of buffer space since credit for a transmission is only granted when space actually becomes available. Unfortunately, since storage space promised must really be available at the destination, smaller window size per connection may be required. Roundtrip time (at least for flow control credits) is also increased because destination processing of the data is included. Both these effects may reduce throughput.

An optimistic buffer allocation policy may allow higher throughput by returning a window size larger than the buffer space actually available. As long as transmission and consumption procede "smoothly," the receiver can provide less

buffer space than the window size returned without having to discard packets. This essentially uses the storage space in the network transmission path to provide the remaining space. The difficulty in this scheme is that promised space may not actually be available when new transmissions arrive. Some packets must then be discarded and subsequently retransmitted.

5.2 Optimistic Buffer Allocation Strategy

Evaluating the performance of the optimistic strategy involves determining the fraction of successfully arriving packets that will be discarded given the buffer space available. Another simple queuing model serves this purpose. Figure 20 shows a queue size limited single server system. Let the queue size, Nbuf, be the buffer space available, and the mean service rate, u, be the consumption rate of the receiving process. The mean arrival rate, A, equals the transmission rate of new packets allowed by flow control and retransmission constraints. We wish to find Pfull, the probability that all Nbuf buffers are full. In the steady state, this is the probability that an arriving packet finds the queue full and hence also the fraction of all arriving packets that must be discarded.

Assuming exponentially distributed interarrival and service times and defining RHO = A/u, Pfull is given by [Kleinrock75]:

FIGURE 20 QUEUING MODEL OF DESTINATION BUFFER SPACE LIMITATIONS

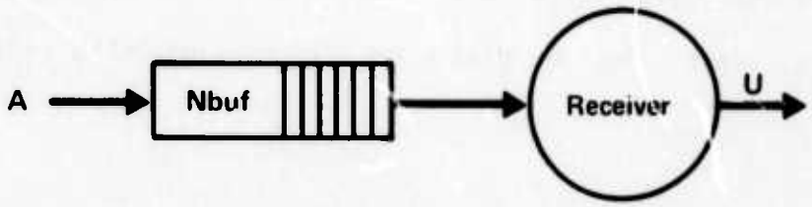
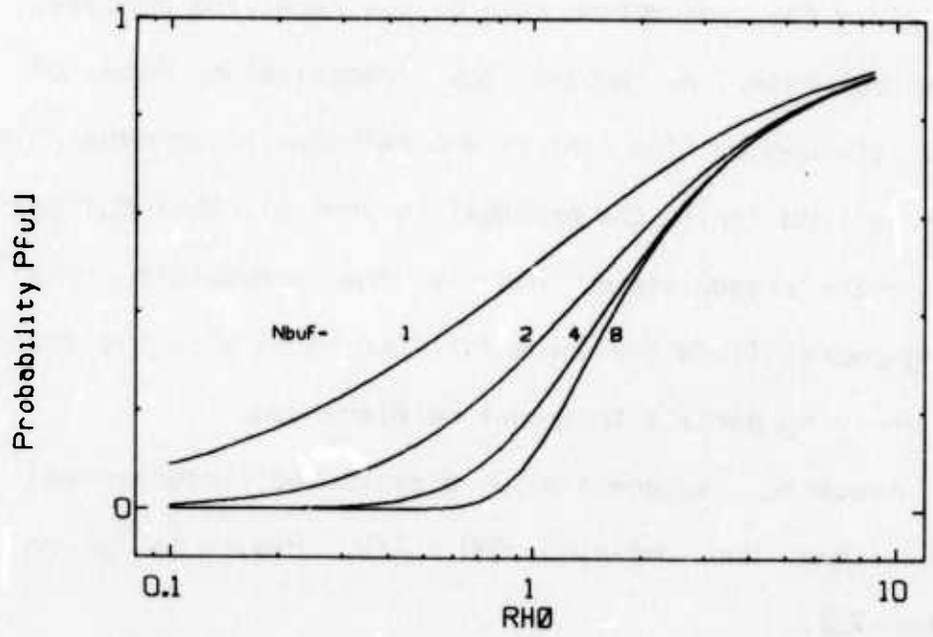


FIGURE 21 PROBABILITY OF DISCARDING AN ARRIVING PACKET, Pfull, vs. RHO = (production rate)/(consumption rate) FOR VARIOUS BUFFER SIZES Nbuf



$$Pfull(Nbuf,RHO) = (1-RHO) \cdot \frac{RHO^{Nbuf}}{1 - RHO^{Nbuf+1}} \qquad (12)$$

Figure 21 shows Pfull versus RHO for various values of Nbuf in this M/M/1 system. Several observations can be made:

For RHO>>1, Pfull approaches (RHO-1)/RHO, indicating that most arriving packets will be discarded regardless of the size of Nbuf. In this case, the sender is transmitting packets at a rate much greater than the receiver can accept them, and throughput is receiver rate limited. By acknowledging packets successfully received (but not yet processed), retransmissions can be reduced. This reduces line utilization but still leaves the receiver with all buffers full regardless of the size of Nbuf. A better solution involves reducing the window size (and Nbuf) to limit the sender's transmission rate.

For RHO<<1, figure 21 shows that Pfull approaches zero. Very little buffering is necessary for a fast receiver. This situation is typical of a fast process serving many slower sources, and buffer pooling techniques may be advantageous [Chu74].

For RHO=1, Pfull=1/(Nbuf+1) in the M/M/1 system. For more realistic distributions with smaller variances, Pfull drops

more quickly as the number of buffers increases. As remarked at the beginning of this section, nearly constant interarrival and processing times reduce Pfull to zero for small numbers of buffers. However, periodic scheduling in multiprogramming systems causes essentially bulk arrivals and processing: When the sender is scheduled, a burst of traffic is generated which accumulates at the destination until the receiver is scheduled. If scheduling intervals are large compared to roundtrip times, increased window size and buffer allocation may be necessary for high throughput.

5.3 Results

The preceding analysis determines the additional throughput degradation TPbuf due to destination buffer limitations. With conservative strategies, no packets are discarded due to lack of buffer space since credits are only returned when space is actually available. However, flow control allocations are directly tied to buffer availability, possibly resulting in smaller window size and longer roundtrip times which both reduce achievable throughput (cf section 4).

With optimistic strategies, window sizes larger than the available buffer space are allowed, increasing achievable throughput or reducing buffer storage required. However, some arriving packets may have to be discarded if promised space is

not actually available. Throughput degradation resulting from overoptimism aquals:

$$TPbuf(Nbuf) = 1-Pfull \quad (13)$$

Pfull depends on the ratio and smoothness of production and processing rates as well as on the buffer space available. When process scheduling delays are significant (compared to roundtrip times) as in multiprogramming systems, the conservative strategy with guaranteed buffer availability may be necessary to reduce discard rates and hence retransmission cost to an acceptable level.

Combining equation 13 with results from previous sections, the maximum achievable throughput for PAR protocols including the effects of overhead, retransmission, flow control, and destination buffer storage limitations is:

$$TPmax = TPoh \cdot \min(TPretrans, UT) \cdot TPbuf \cdot B \quad (14)$$

Buffering for rate smoothing purposes is an example of the often discussed producer-consumer problem [Dijkstra68, Coffman73]. In the distributed environment of computer network communication, the producer and consumer may be more loosely coupled than in centralized systems. Using the conservative strategy causes the sender (producer) to be blocked from new transmissions when the allocated space is full (the normal situation in tightly coupled centralized systems). Using the

optimistic policy allows the sender to transmit new packets which the receiver may have to discard if all buffers are full (not normally allowed in centralized systems).

6. SEQUENCING

The qualitative performance goal of delivering packets in order is relatively easy to implement as described in chapter II by including a sequence number in the header of each packet (SPAR protocol). However, introducing sequencing significantly complicates quantitative performance analysis because delay for neighboring transmissions can no longer be assumed independent.

In particular, a packet may arrive successfully at its destination before one of its predecessors because the transmission medium does not always deliver packets in the order submitted (cf section I-2). Briefly, this is due to alternate routing and line errors followed by retransmissions within a PSN, or to damage or complete loss of an earlier packet. If the protocol allows packets to be transmitted in excess of buffer space available at the destination (cf section 5), packets arriving out of order may have to be discarded, requiring retransmission and degrading throughput as we show in section 6.2. Even when buffer space is available, the packet is not accepted (delivered to the receiving process) or acknowledged until all its predecessors have successfully arrived. This causes an explicit dependence among the transmission delays for neighboring packets.

6.1 Increased Roundtrip Delay

We can derive a rough estimate of increased delay when sequencing is required by using basic probability arguments similar to those used in deriving equations 4 and 5. In this case, we wish to find the time delay distribution for a packet and all its predecessors to have arrived.

For purposes of analysis, a conceptual change in the acknowledgement strategy proves expedient. Normally an Acknowledgement is returned for an arriving packet only after all predecessors have arrived. Instead, suppose an Acknowledgement is returned immediately for all successfully received packets, regardless of arrival sequence, but the Acknowledgement is not accepted ("believed") until Acknowledgements for all previous packets have arrived. The roundtrip time from first transmission to accepting an Acknowledgement is the same for both schemes. For the second scheme, G(t) from equation 4 already gives the roundtrip delay time distribution for each Acknowledgement to return. Only at the last stage in the analysis does the dependence on previous transmissions come into play.

Let Tint be the (fixed) interval between transmission of sequential packets (ignoring retransmissions). Let H(t) be the desired roundtrip delay cumulative distribution including the additional delay incurred when packets arrive out of order and must wait for some missing predecessors to arrive.

$$H(t) = \text{Prob}\{\text{ACKs for packet i and all predecessors have arrived by time t}\}$$

$$= \text{Prob}\{\text{ACK i arrives by time t}\} \cdot \text{Prob}\{\text{ACK i-1 arrives by time t}\} \cdot \text{Prob}\{\text{ACK i-2 arrives by time t}\} \cdot \ldots$$

$$= G(t) \cdot G(t+Tint) \cdot G(t+2 \cdot Tint) \cdot \ldots$$

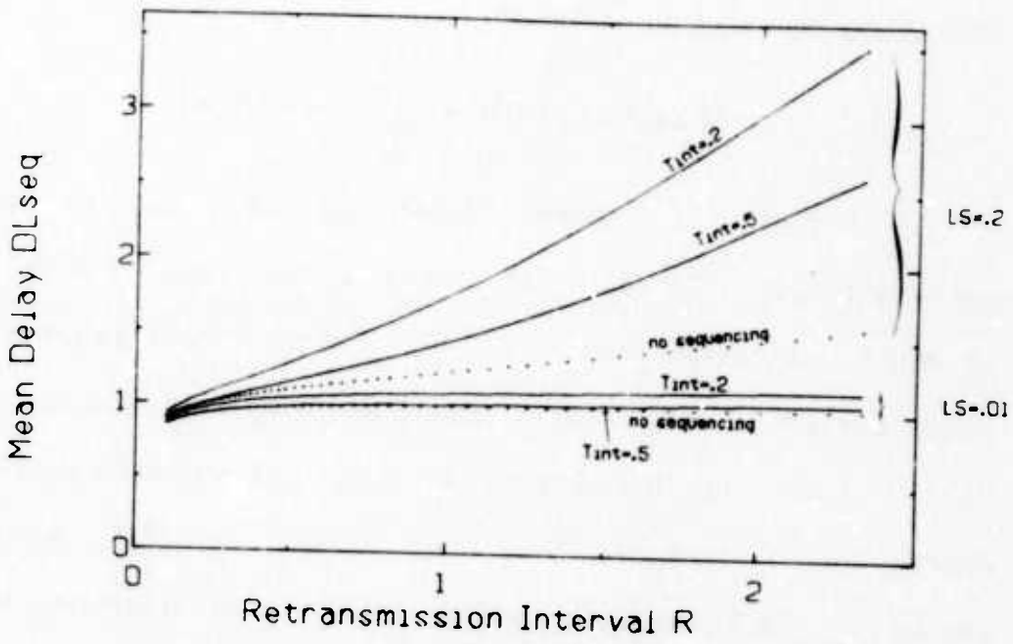
$$= \prod_{j=0}^{\infty} G(t+j \cdot Tint) \qquad (15)$$

Using equation 6 we can immediately write the mean delay including sequencing, DLseq as:

$$DLseq(R,F,Tint) = \int_0^{\infty} [1-H(t)]\, dt \qquad (16)$$

Figure 22 shows DLseq as a function of the retransmission interval R for several packet loss probabilities LS and transmission intervals Tint. Delay without sequencing is shown dotted for comparison. The underlying transmission medium delay, f(t), is the Erlangian distribution with mean=1 and degree k=16 from section 3.3. The spread of f(t) represents variation in transmission medium delay due to alternate routing and PSN internal error recovery. The loss factor and retransmission interval account for lost or damaged packets and end-to-end error recovery.

FIGURE 22 MEAN DELAY INCLUDING SEQUENCING, DLseq, vs. RETRANSMISSION INTERVAL R FOR SEQUENCING PROTOCOL



LS = PROBABILITY OF PACKET LOSS

Tint = TRANSMISSION INTERVAL OF NEW PACKETS

TRANSMISSION MEDIUM DELAY DISTRIBUTION f(t) IS IS ERLANGIAN (mean = 1, k = 16)

Results:

Small transmission intervals Tint (rapid transmission rates) increase delay required to resequence packets since it is more likely that closely spaced packets will arrive out of order. For low packet loss probabilities (LS<<1), the increase is small for typical f(t) with low variance. However, as network traffic increases, the variance of transmission time also increases [Naylor73], and resequencing delays may become more significant.

For larger LS, a significant fraction of packets are delayed by one or more retransmission intervals R. When sequencing is required, this affects the preceding packet delays as well, amplifying the increase of delay with R noted in section 3 by a large factor depending on Tint.

Reducing Retransmission Rate:

When a packet is damaged or lost in a sequencing protocol, not only the Acknowledgement for that packet, but also for all subsequent packets within the window size Nwin will be delayed by at least a retransmission time R. Hence it is likely that the retransmission time-outs for all the other pending packets in the window will also expire, and all Nwin packets will be retransmitted following the faulty one. Some protocols attempt to avoid this amplification of retransmissions by

suppressing retransmission of all but the first timed-out packet.

Such single packet retransmission protocols then operate in a bimodal fashion: In "normal" mode (no retransmission time-outs have occurred), they transmit with window size Nwin and throughput derived in section 4 above. In "error" mode, they retransmit only the single timed-out packet at intervals R until successful. This policy keeps transmission medium bandwidth, retransmissions, and hence cost to a minimum (since Acknowledgements for the other successfully received packets can return before they are retransmitted). But delay is increased and throughput decreased since one of the other (suspended) packets may also have been lost or damaged. For realistic loss probabilities (LS<<1), the savings and performance degradation are minimal, but the protocol implementation may be significantly simpler with such a single packet retransmission strategy.

Returning negative acknowledgements (NACKs) for damaged packets provides another way to reduce retransmission costs and delay. The NACK can stimulate immediate retransmission of the damaged packet before its normal retransmission time would occur. If the second transmission is successful, a positive acknowledgement of the missing packet and its successfully received successors may reach the sender before a full window of packets has been retransmitted. Although the protocol cannot

rely on NACKs for reliability (cf section II-1), NACKs may provide an improvement in efficiency when damage rates are significant.

6.2 Discard Probability

As shown above, protocol performance deteriorates when sequencing is required even if there is enough destination buffer space to hold all out of order arrivals. In this section we examine the additional degradation resulting when out of order arrivals must be discarded due to insufficient buffer space.

Along the lines of section 5, we wish to find the probability that an arriving packet must be discarded, Pdis. This results in throughput degradation by a factor 1-Pdis exactly as in section 5. Pdis is most easily derived by first considering the probability that an arriving packet is <u>in</u> order, with the possible exception of its n-1 most recent predecessors:

$$\begin{aligned} Pinord(n) &= Prob(\text{when packet } i \text{ arrives, packets } i-n,\ i-n-1,\\ &\qquad i-n-2,\ \ldots \text{ have already arrived})\\ &= \int_0^\infty Prob(\text{packet } i \text{ arrives at time } t, \text{ and}\\ &\qquad \text{packets } i-n,\ i-n-1,\ i-n-2,\ \ldots\\ &\qquad \text{have arrived by time } t)\ dt \end{aligned}$$

Note that Pinord(1) is the probability that an arriving packet is in order (i.e. none of its predecessors is missing). Once again, let Tint equal the (fixed) interval between transmission of sequential packets. Let $\hat{G}(t)$ and $\hat{g}(t)$ be <u>one way</u> delay time distributions including loss probability LS and retransmission at intervall R. (We estimate $\hat{G}(t)$ and $\hat{g}(t)$ by the roundtrip delay distributions G(2t) and g(2t).) Then Pinord(n) can be written:

$$\text{Pinord}(n) = \int_0^\infty \hat{g}(t) \cdot \hat{G}(t+n\cdot\text{Tint}) \cdot \hat{G}(t+(n+1)\cdot\text{Tint}) \cdot \ldots \, dt$$

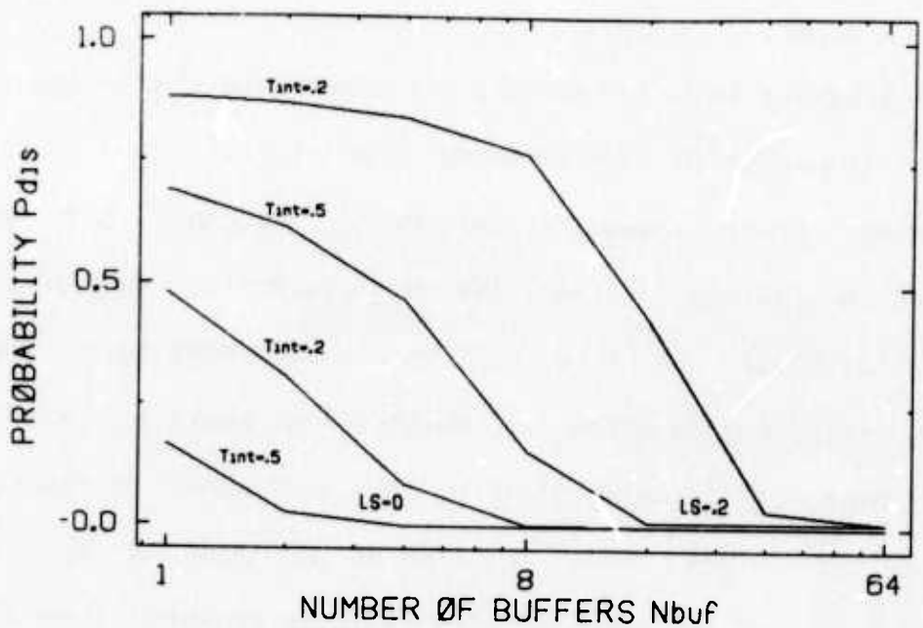
$$= \int_0^\infty \hat{g}(t) \prod_{j=n}^{\infty} G(t+j\cdot\text{Tint}) \, dt \qquad (17)$$

Finally note that with a buffer size of n, the probability that an arriving packet must be discarded is just 1-Pinord(n), the probability that at least one of the packet's predecessors n or more away has <u>not</u> arrived yet:

$$\text{Pdis}(n,\text{Tint}) = 1-\text{Pinord}(n) \qquad (18)$$

Figure 23 shows Pdis as a function of n for several values of Tint and LS. The underlying transmission delay distribution is again assumed Erlangian with mean=1 and degree 16. R is 1.5 in the optimal range determined in section 3. Closely spaced transmissions increase the likelihood of out of order arrival and the discard probability. For small LS, only alternate routing and internal PSN error recovery (single hop

FIGURE 23 PROBABILITY OF DISCARDING AN ARRIVING PACKET, Pdis, vs. DESTINATION BUFFER SPACE FOR SEQUENCING PROTOCOL



LS = PROBABILITY OF PACKET LOSS

Tint = TRANSMISSION INTERVAL OF NEW PACKETS

RETRANSMISSION INTERVAL = R = 1.5

TRANSMISSION MEDIUM DELAY DISTRIBUTION IS ERLANGIAN (mean = 1, k = 16)

retransmission) contribute to out of order arrival rates, and a moderate number of buffers suffices for nearly all resequencing needs. For higher LS, longer end-end retransmission delays contribute significantly to less ordered arrivals, and more buffering is required to avoid discarding packets. Except for very long transmission intervals (low throughput) or low variance transmission medium delay distributions with small LS, a significant fraction of packets are discarded with the "simple" sequencing protocol strategy of accepting only the next packet in order.

Equation 18 and figure 23 are based on a simple analytic model of transmission medium delay characteristics. Although this model proves adequate for mean throughput and delay analyses in previous sections, the relative arrival sequence of packets discussed in this section is much more sensitive to small correlations in delay of neighboring packets, and the exact shape of the delay distributions that occur in real PSN. Therefor the values shown in figures 22 and 23 are more representative of the shape of effects to be expected than their exact values.

In fact Pinord(n) is the sort of performance measure that proves exceptionally difficult to derive exactly for a detailed PSN model. Even the mean interpacket arrival times under the assumptions of a fixed path and no reordering proved a formidable problem [Fultz72]. Fortunately Pinord(n) is easily

obtained empirically by comparing arriving packet sequence numbers with the current expected sequence number (ESN), and tabulating a histogram of differences. Such information has not been recorded until recently by Forgie who derived a simpler global out of order percentage from his data [Forgie75].

7. PACKET SIZE

Although discussed last, packet size selected by the protocol by no means has the least impact on performance. The primary result of varying packet size is to vary the basic transmission medium delay f(t). Transmission delay in a PSN has a large component proportional to packet length because the transmission time on each hop (between switching nodes) is equal to packet length divided by bandwidth [Metcalfe73]. Therefore shorter packets mean lower per packet delay, with ensuing effects on retransmission, flow control, and buffering. Large packets are also undesirable because they require a bigger share of protocol buffer space and a larger slice of available transmission bandwidth, raising the question of fairness to other processes sharing these resources.

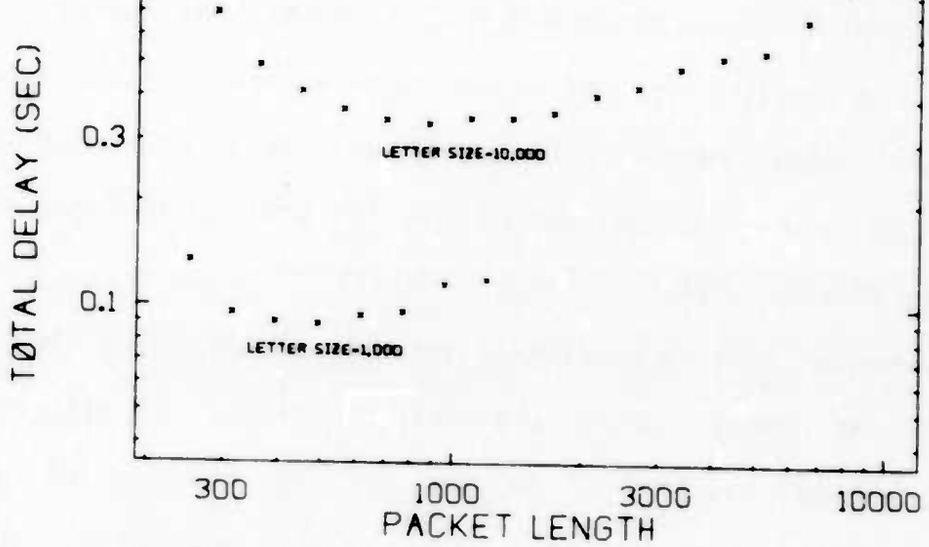
The main counterforce to sending short packets is the increase in overhead. Since header and control information is normally fixed length, a larger portion of available bandwidth is taken up with overhead as packet size shrinks [Kleinrock74]. Furthermore, larger processing overhead and space for associated linkage and state information is required by the protocol for each data bit transferred. Maximum throughput attainable decreases with shorter packet lengths, and cost increases since the number of packets or total bits required to transmit a given amount of data increases.

Let a letter be an independently meaningful chunk of data, i.e. the amount of data necessary for the receiver to begin processing. This may vary from a single character for some text editing applications, to a large file for a compiler. The total delay to transmit a large letter first drops with decreasing packet size because of lower per packet delay, but then begins to rise again with shorter packets because of increased overhead. Figure 24 illustrates this for a representative set of parameters but no packet loss (LS=0).

The probability that transmission errors will occur also rises with packet length, giving an upper limit to achievable throughput and an optimal packet size for maximum throughput as shown in [Metcalfe73].

Hence there is no single optimal packet size for an interprocess communication protocol to select in all cases. Rather, optimal packet size depends on the balance of user requirements for delay, throughput, cost, and letter length [Opderbeck74]. Short data transmissions where each transaction may be independently processed can take advantage of reduced delay in using shorter packets, but incurring higher costs and lower throughput. Real-time traffic requiring moderate delay and good throughput for moderate letter sizes should use moderate packet sizes. Minimum cost or optimal throughput users willing to tolerate longer average delay should use longer packets. As transmission medium bandwidths rise and error rates drop, the impact of packet size will be lessened.

FIGURE 24 TOTAL DELAY vs. PACKET LENGTH FOR VARIOUS LETTER SIZES



NETWORK BANDWIDTH = 50 kb/sec
NUMBER OF HOPS = 5
HEADER LENGTH = 200 BITS
PACKET LOSS PROBABILITY = 0

Packet switching networks have their own similar reasons for selecting packet sizes for internal transmission. The units of information accepted from network users (e.g. ARPANET messages) may be larger or smaller than these internal packet sizes. The ARPANET offers a larger input limit (8 kbit messages), choosing to fragment large user submissions into smaller packets for internal transmission to reduce delay and switching node buffer requirements [Crowther75]. TYMNET, on the other hand, collects short user inputs into larger packets for internal transmission to reduce overhead [Tymes71].

Although an interprocess communication protocol in general has no control over such internal PSN transmission decisions, an awareness of transmission characteristics is fundamental to efficient protocol operation as we have seen. For example, delay is not linear with packet (message) sizes above 1 kbit in the ARPANET due to the internal fragmentation of larger submissions mentioned above. Hence use of larger packet sizes by an interprocess communication protocol on the ARPANET becomes more attractive.

Chapter IV

NETWORK INTERCONNECTION

1. INTRODUCTION

As computer networks proliferate, the performance of interprocess communication protocols will receive increasing attention. Chapters II and III provide a basis for the design and analysis of protocols to meet particular reliability or efficiency performance goals.

Another set of questions quickly growing in importance concerns the interconnection of computer networks. Networks are already developing on the basis of geographical coverage, particular types of service offered, and organizational coverage. Users desiring access to multiple areas, services, or organizations will need to communicate over many of these networks as easily as possible. The general computing power and unique computing resources available on different networks can most efficiently be made available to people and computers attached to other retworks by interconnection of networks.

We shall adopt the term Gateway for the interface between interconnected networks as used in IFIP Working Group

(1) IFIP is the International Federation for Information Processing. Working Group 6.1 is the Internetwork Working Group.

6.1 (1) (INWG) [Pouzin73, Lloyd75a, Cerf73]. As a first model of network interconnection, we can consider the Gateways as "supernodes" in a "supernetwork." Individual local nets are just the "lines" that connect Hosts to supernodes, and supernodes to each other. The unusual aspect of this supernetwork is that each line may require a different communication protocol, so the supernodes must implement the correct local net protocol for the nets they interface. Even circuit switched nets could serve as lines in the supernetwork, although much of the following discussion applies most directly to packet switching networks (PSN's). While this supernetwork model is by no means the only sort of interconnection possible, it provides a simple introduction to the issues involved in interconnecting heterogeneous computer networks.

Even for identical networks, such interconnection is not a trivial problem. As a minimum, common addressing techniques are needed so any user on any of the interconnected networks can uniquely specify any other users. Global addressing can be achieved by expanding the address space available on each local net and changing previously identical local net addresses (highly inconvenient to the users involved), by concatenating partial path addresses into a complete path specification, or by instituting a hierarchical address space (e.g. net, local address) with necessary alterations to routing algorithms. Section 2 below considers these addressing and routing alternatives.

In general, communication formats and protocols may differ between networks to be connected. Hence the primary function of a Gateway is to translate between formats and protocols used in each local net. This raises the difficult problem of choosing an appropriate level of interconnection, discussed in section 3 below.

As part of the translation process, Gateways must deal with varying maximum packet sizes in the local nets connected. When a packet arrives that is too large for the next local net, the Gateway must fragment the packet before forwarding it through the next local net. These fragments may be reassembled as they leave the next local net, or allowed to procede independently to their ultimate destination.

Since Gateways are nodes in the supernetwork formed by connecting individual local networks, they must also support typical node-to-node communication functions. Flow control and buffer allocation algorithms are necessary to limit peak loads and to share resources fairly. Access control, accounting, and performance monitoring are specially important to Gateway nodes since relatively independent local networks with potentially sensitive political and administrative concerns are involved [Kuo74, Kuo75]. In section 4 below we consider several of these additional Gateway functions.

Other typical node functions such as error detection, duplicate detection, sequencing, and retransmission may be

performed on a hop-by-hop (Gateway-Gateway) and/or end-end basis. Some of these, such as retransmission, may be desirable hop-by-hop (at least in high loss rate nets), while others such as sequencing may degrade performance if employed on each hop as discussed in section 3.

An important goal of network interconnection strategies is to require as little alteration as possible to the individual networks connected. Expressed in some form by many authors, this goal can be summed up in the following principle:

> Local Net Independence Principle: Each local net shall retain its individual address space, routing algorithms, packet formats, protocols, traffic controls, fees, and other network characteristics to the greatest extent possible.

Some important motivations for this goal are

(1) Local nets have a large investment in existing implementations which can not be replaced inexpensively.

(2) Most net traffic will continue to be local net traffic and it is unfair for all users to suffer the disruption of service and increased cost of a new implementation that only serves a minority of users.

(3) Even if technically desirable, political, economic, or administrative constraints may make changing to global standards impossible.

(4) From a practical viewpoint, cooperation will be more likely and completion faster if fewer changes are required.

On the other hand, some global agreements are clearly necessary for meaningful communication, for example, standard addressing techniques so users can refer to each other unambiguously, and common formats so arriving messages can be correctly interpreted. The goal is to implement such standards on top of existing local net functions, achieving independence and universality at the same time.

Reference is made throughout the remainder of this section to several existing or planned networks as examples of various points discussed. These nets include ARPANET, CYCLADES, EPSS, TYMNET, ALOHANET, PRNET, UCL, and DCS. Appendix B provides a list of references relevant to each of these nets for the reader wishing background information or to further explore the points raised below.

2. ROUTING AND ADDRESSING

The question of addressing, or how to name all the participants in an interconnected communication system, is intimately related to the question of routing, or how to find paths from source to destination and then choose among them. For example, a single level address space requires each node performing routing to know the correct route to every possible destination independently, while a hierarchical address space allows routing nodes to know correct routes only to destinations within the local "area" and to other areas (although such area routing may not be optimal) [McQuillan74].

A large body of literature exists on routing and addressing for individual networks [Baran64, Fultz72, McQuillan74, Frank71, Farber73], but only recently have the special problems pertaining to network interconnection been addressed [Graham71, Farber73, Belleni74, McQuillan74]. Many of these problems stem from the Local Net Independence goal, i.e. the desire to preserve individual address spaces and routing techniques within each local net. This favors restricting internet functions to Hosts and Gateways rather than implementing them within local nets as we shall see below.

A number of important concepts in addressing have proved confusing due to conflicting use of terms by different authors. Hence before continuing we present brief definitions of the terms to be used in the remainder of this section.

PSN: Network of packet switches that forward packets of appropriate format from a source Host to a destination Host. Various additional services such as sequencing, retransmission, delivery confirmation, etc. may also be available.

Host: Source and destination of packets in a PSN. For routing purposes, all packets to a given Host are going to the "same place" as far as the PSN is concerned. Hosts may be singly connected (to a single packet switch) or multiply connected (to more than one packet switch) in which case optimal Host-Host routing is more complicated.

Communication Control Protocol (CCP): As described in chapter II, this represents the end-end protocol which provides reliable interprocess communication and multiplexes the independent communication streams from many processes within a Host. Examples of CCPs are TCP [Cerf74b, Cerf74c], NCP [Carr70], NCAM [Karp72], and Transport Station [Zimmerman73]. In the simplest cases, there is a one-to-one correspondence between CCP and Host, and a CCP name is synonymous with its Host name. (In ARPANET the Host name is emphasized, and all packets to a Host typically go to a single NCP.)

In general CCP names are independent of Host names. There may be multiple CCPs in a single Host (CYCLADES). A CCP may "move" from one Host to another [Lay73] (although routing tables must reflect the altered CCP/Host correspondence). A single CCP may even encompass several Hosts. (The multi-Host ARPANET installation at BBN approaches this type of application although at a higher protocol level.)

Port: The ultimate source and destination of the communication path provided by a CCP. Each pair of source/destination ports represents a unique communication path (connection, association), so that a single port may have multiple connections to different remote ports.

Process: Processes represent the active computing tasks (jobs, devices, users) in Host computer systems. Processes in different Hosts (or the same Host) wishing to communicate with each other must first acquire ports from their local CCPs. Association of processes and ports in each CCP is completely a local matter, but a number of "well-known" ports associated with particular services at each CCP are useful.

Gateway: The interface between local networks. Although discussed more fully in the next section, it is important to

note that a Gateway may "look like" a packet switch (follow PS-PS protocol with local nets), or a Host (follow Host-PS protocol with local nets).

2.1 Local Net Participation in Internetworking

Having defined these important terms, we return to our initial "supernetwork" model of network interconnection to examine local net functions required to support internet communication. We assume that CCP and Host names are synonymous (one CCP, fixed, per Host) and that Hosts are singly connected. This encompasses a large percentage of practical situations. Subsequently we shall consider complications resulting when these constraints are relaxed.

A source CCP (Host) creates an internet packet containing data and a header with necessary control information for efficient and reliable communication (cf chapter II). The header includes a two level internet destination address of the form (net, local address). This internet packet must be delivered through the CCP's local net to a Gateway for forwarding to the destination net.

Unfortunately, an internet packet may not be suitable for direct transmission by local net protocols. Instead, the entire internet packet must be presented as data to the local net protocol, and wrapped in the appropriate local net header

(and perhaps trailer) with the local net address of the Gateway (see figure 1). This concept of embedding has appeared frequently in work by members of INWG [Cerf74a, Cerf74c, Pouzin73, Pouzin74a]. At a Gateway, the local net "envelope" is removed and the internet packet extracted. The internet destination address can then be used to route the packet to another Gateway, or to the final local Host, and the internet packet is re-embedded for transmission through the next local net.

This embedding strategy complies fully with the Local Net Independence principle since no changes at all are required in local net addressing or routing. In fact local nets are completely unaware that they are carrying internet traffic.

The disadvantage of such strict adherence to local net independence is that source Hosts must generate the local net address of the first Gateway. Then each Gateway must not only choose the next net, but also the specific next Gateway and specify its local net address. This local address (and the rest of the local net header) must travel with the internet packet through each local net, increasing overhead. To eliminate these disadvantages, it is possible to alter local net operation to interpret the internet header and address directly, avoiding the need to embed internet packets. To this end, part of the internet header may be reserved for local net functions, while the remainder is used by the CCP and Gateway for internet

FIGURE 1 EMBEDDING INTERNET PACKET IN LOCAL PACKET

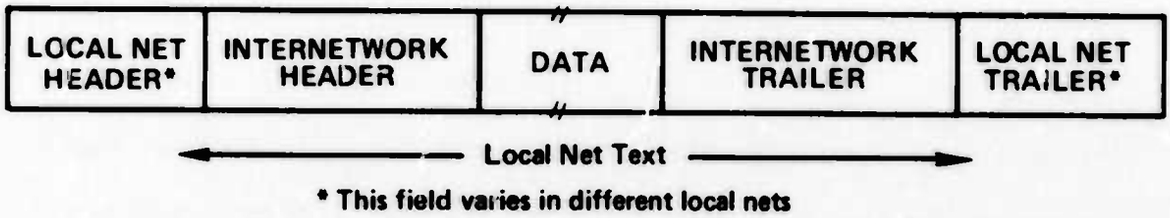
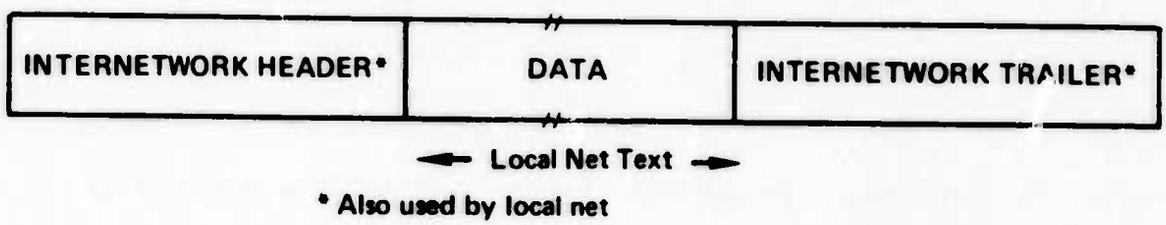


FIGURE 2 OVERLAPPING INTERNET AND LOCAL NET PACKET HEADERS



functions. Such an overlapping packet format has been proposed in recent INWG documents [Cerf75, INWG74, INWG75a]. (see figure 2)

Direct internet packet interpretation would require a substantial change in established nets and is probably unacceptable. To minimize changes to Host software, only an escape or type field may be added to the existing formats and protocols, so that local and internet packets can be differentiated and treated appropriately. Of course new networks may be designed to understand a hierarchical address format, or to implement multiple packet types from the start.

Even if local nets do understand the internet address, requiring them to route internet packets based on the internet destination represents a significant additional burden. When only one Gateway exists, routing in the local net is trivial. The "central office" interconnection model [McQuillan74] where all internet traffic is routed to a local central office, then between central offices of different nets (by special trunk lines), and finally to a local destination, also presents simple local net routing of internet traffic.

For reasons of reliability as well as efficiency, multiple Gateways (or central offices) connecting networks are desirable in supernetworks of nontrivial size [Weber64]. As soon as multiple Gateways exist, local net routing and internet routing lose their independence. The local net cannot choose

the "best" local Gateway (by whatever criterion) unless it knows the cost of the remainder of the possible routes from Gateways to final destination. This represents another significant breach of local net independence since local nets require global routing information, and argues strongly that next Gateway selection be performed in the source Host and in Gateways, rather than in each local net.

In loop and broadcast networks, each packet transmitted is available to every node on the network, but the Gateway selection problem still arises. Either the source must specify the particular local Gateway to accept the packet in addition to the final destination address, or the local net must allow Gateway nodes to capture packets on the basis of global destination addresses. The latter alternative again involves local nets in internet routing decisions.

Pierce (1972) has proposed a multi-level hierarchical loop system where routing of internet packets by network interface nodes (Pierce's "C boxes") is particularly simple. Each interface node connects exactly two loop nets, normally at adjacent levels in the hierarchy. If a packet in a local net is destined for a different local net, the interface passes it to the higher level "regional" net. If a packet in the regional net is destined for the attached local net, the interface passes it down to the local net. The same matching test is performed at interfaces between regional nets and the "national" net at

the top of the hierarchy. Packets follow an essentially fixed path up and then down the hierarchy of nets, although Pierce has suggested alternate routes for failure recovery and direct connections between local nets exchanging high volumes of traffic.

Graham and Pollack (1971) have presented another system for simplifying internet routing in a more generally connected (non-hierarchical) system of loop nets. In their proposal, addresses of all networks are carefully constructed so that the Hamming distance (1) between addresses corresponds to the path length between the corresponding nets. When a packet enters a network interconnection node, its destination address is compared to the addresses of the two nets connected, and the packet is routed to the net giving the smallest Hamming distance.

The drawbacks of this scheme are the length of addresses required to provide a successful distance comparison, and the sensitivity of addresses to topology. Address length is proportional to the number of nets in the system, n, rather than log n as with normal addressing. Any change or addition to network topology requires a new address construction, often resulting in changes to many existing addresses.

---

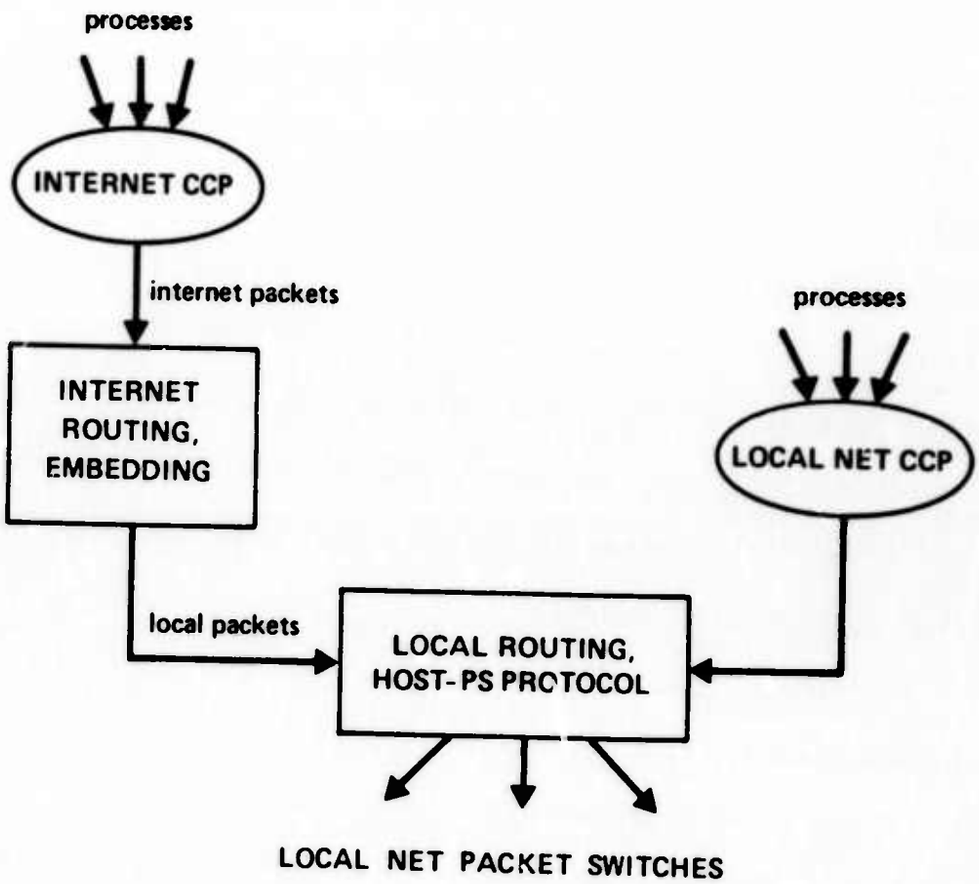
(1) The Hamming distance between two binary numbers is the number of bit positions in which they differ.

Both of these loop interconnection schemes are aimed at shortest path routing and are not readily adaptable to provide routing on other criteria such as bandwidth, delay, or cost (cf section 2.2). Routing is essentially fixed rather than adapting to varying network performance, although some recovery from total connection failures has been provided.

The need for Gateways to perform a routing function is not surprising, while the need for source Hosts to do so is. This need is a direct consequence of the multiplicity of Gateways in a local net. Remembering the analogy between a supernet and a local net, a supernetwork with multiple Gateways parallels a local net with multiply connected Hosts (Hosts connected to more than one packet switch). Hence we will call such networks multiply connected networks and note that every Host in a multiply connected network is thereby (locally) connected to multiple Gateways.

Currently, multiply connected Hosts are a rarity (impossible on the ARPANET) so it is unusual to think of Hosts (CCPs) making routing decisions and exchanging routing data. For Hosts (or local nets) engaged in internetwork communication, two levels of routing are required as observed in [Belloni74] (see figure 3). Internet packets and addresses are generated by the internet CCP for each connection. Then the internet routing level selects a Gateway based on the internet destination, and attaches the local net address of the Gateway. Finally, the

FIGURE 3 TWO LEVELS OF HOST ROUTING



local net routing level selects a packet switch based on the attached local net address. The packet is then ready for transmission over the appropriate line using the Host-PS protocol. Either routing level may be a single fixed choice if either the network or the Host is singly connected. In the most general case, a single Host (CCP) may be connected to multiple networks, in which case the internet routing level selects among the several local net routing levels.

When multiple CCPs reside in a Host, they may share the routing levels (see figure 3). A CCP generating only local traffic requires only the local routing level, while an internet CCP requires both levels.

2.2 Routing Data Structures and Control Strategies

Having established the general outline of internet routing, we now take a closer look at the data structures and information exchanges necessary to support various routing systems. Routing data structures maintain the information on possible paths and their relative costs that is needed to make routing decisions. Routing control strategies define how this information is obtained, updated, and used to make routing decisions.

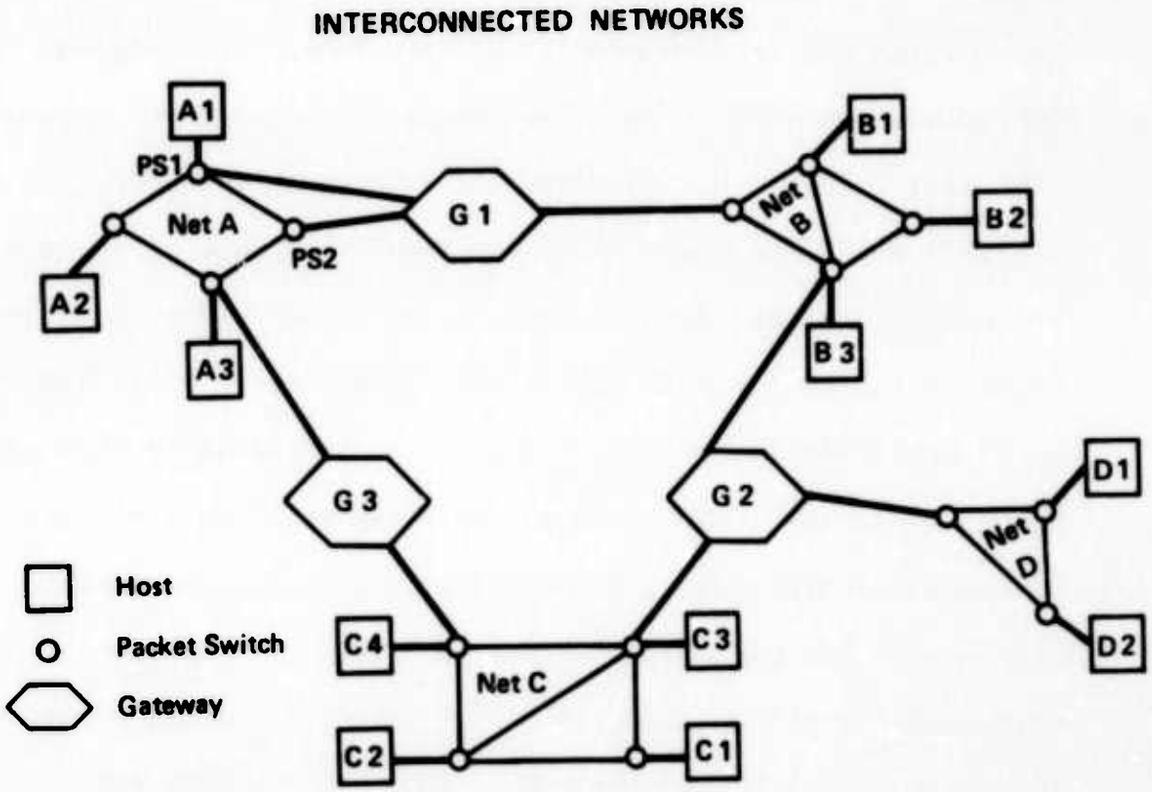
Data Structures

The basic data structure of a routing system is typically called a routing table (RT). RT contains a row for each possible destination, and a column for each route leaving the node. The entries of RT are the "costs" of reaching the given destination by the given route, including special entries meaning the destination is local (directly connected by the route), or unreachable by a given route. Normally a routing algorithm selects the route with minimum cost for a given destination, or gives up (and possibly returns an error message) if the destination is unreachable by any route.

Such routing tables can model a large class of routing algorithms if the cost is suitably defined [Belloni74]. In general, there may be several routing objectives (minimum delay, maximum bandwidth, shortest path, minimum charge, avoid certain nets, etc.), each with its own cost function and RT. Presumably arriving packets indicate in some way by what criterion they wish to be routed.

Figure 4 presents an example of interconnected networks, and shows a routing table for Gateway G1 between nets A and B. The five routes from G1 include local routes to packet switches in nets A and B, and forwarding routes to other Gateways G2 and G3. In this example, the cost measure is path length, resulting in shortest path routing.

FIGURE 4 GATEWAY ROUTING TABLE FOR A SINGLE LEVEL ADDRESS SPACE



ROUTING TABLE* FOR GATEWAY 1

Destinations	Routes: PS1	PS2	B	G2	G 3
A1	1	4	∞	large	
A2	2	3	∞		
A3	3	2	∞		
B1	∞	∞	2		
B2	∞	∞	3		
B3	∞	∞	2		
C1	∞			4	5
C2				4	4
C3				3	4
C4				4	3
D1				4	6
D2				4	6

Destinations: Local Destinations (A1, A2, A3, B1, B2, B3); Remote Destinations (C1, C2, C3, C4, D1, D2)

I: Local Routes — II: Forwarding Routes — III: ∞ — IV: large

* Cost measure is path length

∞ = Unreachable

Dividing the rows and columns of RT on the basis of "local" or "remote" produces four regions as shown in figure 4. Each region has a characteristic structure, simplifying the implementation of RT. The local routes to each net are normally the best routes to all the local Hosts on that net, giving region I of RT the simple structure shown in figure 4. Region II routes are not normally used, having a much higher cost than local routes to the local Hosts, but if a local Host becomes unreachable through its local net, it may be reachable through a longer internet route. A local route can never be a route to a remote destination (Hosts do not forward traffic), so region III consists of "unreachable" entries. Region IV represents the most significant portion of RT for internet routing since the best path to remote Hosts may go through various Gateways.

An important argument in favor of a hierarchical internet address space concerns the size of routing tables. In a hierarchical system, each Gateway (or internet Host) is constrained to know only about routing to Hosts in its local nets, or to other nets. All Hosts on a remote net are equivalent for routing purposes. Hence RT may be divided into internet and local routing levels, with the total number of rows reduced to the number of nets (internet level) plus the number of local Hosts (local level) (see figure 5).

Routing with a hierarchical address space is optimal from source to destination net Gateway, and from destination net

FIGURE 5 GATEWAY ROUTING TABLES FOR A HIERARCHICAL ADRESS SPACE
(See FIGURE 4 for network diagram)

INTERNET ROUTING TABLE FOR GATEWAY 1

Destination Nets	Routes A	Routes B	Routes G2	Routes G3
A	+	∞	large	
B	∞	+		
C	∞		3	3
D			3	5

LOCAL NET ROUTING TABLE FOR GATEWAY 1

Destinations Local	Local Net A PS1	Local Net A PS2	Local Net B
A1	1	4	
A2	2	3	
A3	3	2	∞
G3	3	2	
B1			2
B2	∞		3
B3			2
G2			2

+ = Local,
∞ = Unreachable

Gateway to destination Host, but may not be fully optimal since RT cannot distinguish routes on the basis of the destination Hosts's location within the destination net. For example, in routing packets to Host C3 of figure 4, Gateway G1 thinks routes through G2 and G3 are equally good because G1 does not know the internal structure of net C.

With a single level address space, every Host requires a row in RT. The total number of rows is equal to the total number of Hosts in all nets which is probably unacceptable for even moderate sized supernets. Furthermore, each Gateway requires information about routing within remote networks, violating the local net independence principle and requiring more information to be exchanged by adaptive routing algorithms (see below). The main advantage of a single level address space is that routing may be optimal since full information is potentially available.

Another consequence of hierarchical addressing concerns the determination of unreachability for remote destinations. Since RT contains a single row for each net, it is impossible to determine the reachability of a particular remote Host so long as its net is still reachable [McQuillan74]. If the remote net has become partitioned, or the remote Host has died, only the final Gateway will have this information and be able to discard a packet destined for the unreachable destination (and possibly return an error message).

Control Strategies

Another important aspect of the routing system is the means by which cost entries are computed and updated. McQuillan (1974) has described four general control strategies. In non-adaptive or deterministic routing, costs are computed once and never changed (or at least very rarely, only in response to major system failures). The other three classes are adaptive routing strategies, isolated, distributed, and centralized. In isolated routing, RT entries are periodically updated only on the basis of traffic behavior observed at the node. No routing information is exchanged with other nodes in either isolated or deterministic routing.

Centralized routing involves one (or more) routing centers which collect traffic information from all nodes, and generate RT updates for all nodes (TYMNET is an example). Finally, distributed routing is based on the regular exchange of routing data between adjacent nodes (as in ARPANET).

Any of these routing strategies could conceivably be used for internetwork routing. McQuillan presents an extensive discussion of each class primarily in a single network context. Below, we discuss the application of these results to network interconnection.

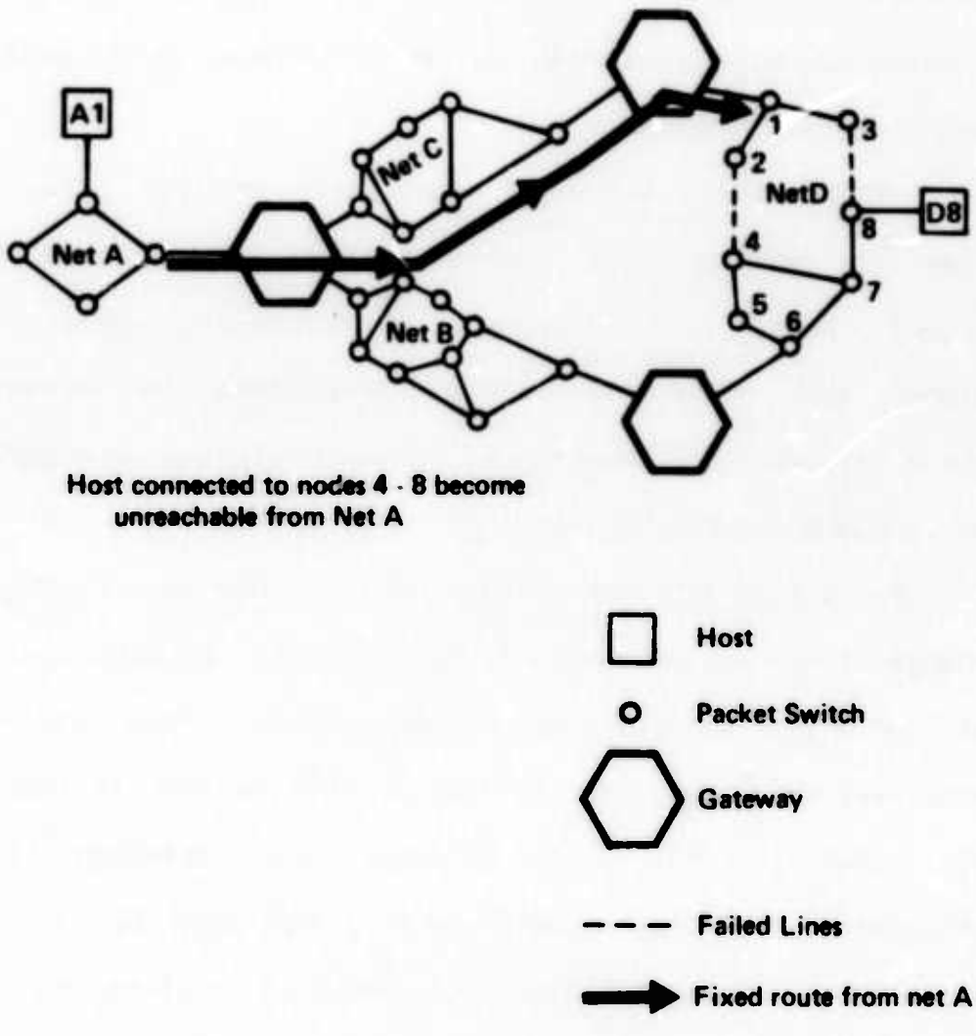
Deterministic routing provides the simplest implementation, but is overly sensitive to failures (fixed routing), or very inefficient since it does not adapt to

changing traffic loads and other conditions. A particularly unfortunate situation can occur with fixed routing when a failure occurs which partitions a local net, or when a Gateway fails. The fixed routing algorithm declares some destinations unreachable because the (fixed) path is broken at some point, while in reality the destinations may still be reachable through another Gateway (see figure 6). With fixed routing, overhead for extended connections may be reduced by assigning a short name at the beginning of each connection, and setting up connection tables at each intermediate node on the (fixed) route. These tables associate the correct outgoing route with the short name in each arriving packet (TYMNET TYMSAT [Tymes71] or Bell ESS [Ewin70]). Packets carry only the short connection name or logical line ID rather than the full destination address.

Isolated adaptive routing is also very inefficient since it must keep "probing" alternate routes to detect changes in net behavior and adapt accordingly [Baran64]. If the probing is slow, then bad paths will persist for a long time, and if the probing is fast, a large portion of traffic is purposely routed on non-optimal paths which may lead to network congestion problems.

Centralized routing algorithms concentrate the RT update calculations at a single center with potentially full information to compute optimal routes. TYMNET and PRNET use

FIGURE 6 HOST FALSELY DECLARED UNREACHABLE DUE TO FIXED SUPERNET ROUTING



centralized routing with some distributed failure recovery procedures in PRNET. Unfortunately, centralization is accompanied by increased sensitivity to failure, high processing requirements that typically rise with the third power of the number of nodes [McQuillan74], and higher loading of communication lines near the routing center with incoming data and outgoing updates. Political and administrative considerations also make a central internet routing center unattractive. (Who will own the equipment, pay for operations, determine control parameters? Do the individual nets want to trust a central authority?)

Distributed routing algorithms provide the best reliability (adaptation to failed components) and efficiency [Baran64, Fultz72, McQuillan74]. More recently, Agnew (1974) has shown that distributed routing algorithms can determine optimal routes, and Naylor (1975) has presented an algorithm that is guaranteed to be loop-free. One caveat here is that when the routing algorithm itself fails (routes incorrectly, or exchanges incorrect routing information), the effects on the rest of the network can be disastrous. Very stringent precautions have been taken to prevent such errors in ARPANET IMPs [McQuillan74], but Gateways and internet Hosts participating in internetwork routing may not be so well safeguarded. Deterministic and isolated routing tend to localize the effect of routing failures since no routing data is

exchanged. Centralized systems would presumably be an order of magnitude more reliable than the individual nodes in a distributed system. Hence catastrophic global routing failure is most to be feared in a distributed internet routing system.

Source Routing

Another routing strategy that does not conveniently fit under any of the above classes is source routing where the source of internet packets specifies the complete internet route. When the entire route accompanies each internet packet, no routing decisions or tables are required at Gateways, but the packet format is complicated and overhead increases. In particular, the packet must carry a varying number of intermediate addresses depending on the path and destination [Farber73]. This overhead may be reduced by setting up a fixed route with connection tables (see above) when a connection is established.

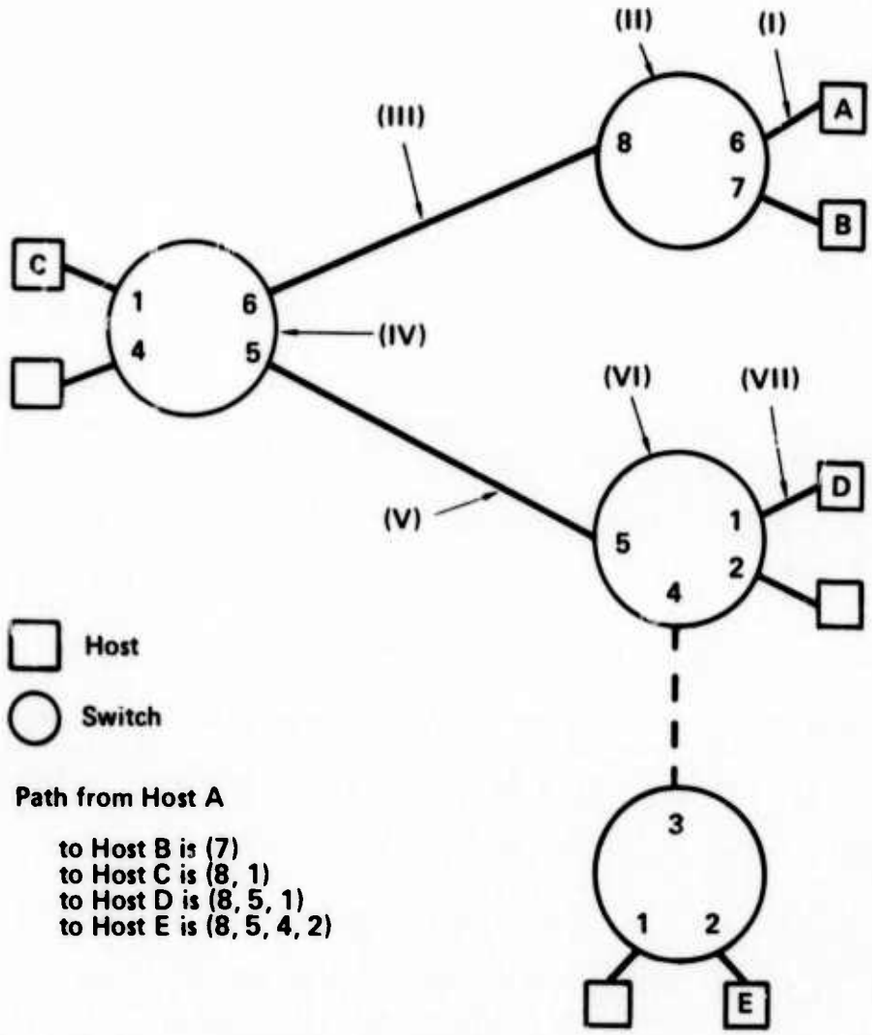
The primary advantage of source routing is the elimination of complex routing responsibilities from intermediate nodes. Instead, responsibility for routing falls on the source nodes which must be able to construct complete routes to any desired destination. Source routing also eliminates the need for global agreement on network names, since the name of each destination becomes equivalent to a path specification for reaching the destination node.

The connection of names and path specifications is particularly apparent in an addressing scheme outlined by Crocker (personal communication). In Crocker's proposal, local networks are represented by a single switch connecting all local Hosts (see figure 7). In addition, some lines from the switch may go to other switches (nets), providing network interconnection. The path to any destination is the series of switch addresses (lines) traversed to reach that destination. A local path is one address long, while paths to remote Hosts require an additional address element for each switch traversed. Figure 7 gives several examples of path specifications.

If each local net (switch) has a globally agreed name, then individual Hosts may be specified by their net name and local Host number, independent of the path(s) available to reach them. However, such global address agreements are not necessary if source routing is used, since any Host may still be addressed by specifying a path to it. This simplifies addition of new networks, or replacement of a single Host by a network, because the new nodes may be addressed by adding one more address element to existing path specifications. For example, if the network containing Host E is attached as shown in figure 7, then a path from Host A to Host E is (8,5,4,2).

With a hierarchical address space and routing by Gateways, addition of a new network requires global agreement on the network name, and insertion of a new row for that network in

FIGURE 7 CROCKER'S ADDRESSING AND ROUTING SCHEME



all Gateway routing tables. With source addressing, only sources accessing a new net (switch) need to know the new topology.

A disadvantage of specifying destinations by their path names is that the "name" of a destination depends on the location of the source. Two different Hosts talking to the same third party may have different paths to and hence different names for the same destination. This situation is similar to dialing a special prefix from an "inside" phone line, or the regular 3-digit prefix from an "outside" phone line to reach the same phone.

Crocker's single switch network model applies most clearly to loop networks and other fully connected nets (broadcast transmission). Farber and Vittal (1973) have proposed a similar source routing approach for interconnection of multiple DCS type loop networks. In addition to specifying its destination, each packet normally identifies its source. Crocker and Farber have described similar means for progressively converting the destination path specification into a return path to the source as the packet traverses successive path elements. Each time the packet leaves a node or switch, the return address of the node is appended to the end of the path specification. Whenever the packet reaches an intermediate destination, the corresponding address is removed from the head of the path specification. For example, figure 7 shows seven

points on the path between Hosts A and D. The path specification at each point is

(i)	(8,5,1)	Start
(ii)	(8,5,1,6)	Return address added
(iii)	(5,1,6)	First destination reached
(iv)	(5 1,6,6)	Return address added
(v)	(1,6,6)	Second destination reached
(vi)	(1,6,6,5)	Return address added
(vii)	(6,6,5)	Final destination reached

When the packet reaches its final destination, the resulting path specification is the path back to the source in reverse order. If the line between switches has the same address (name) in both switches, then the two address transformations described above become a simple cyclic shift (e.g. point iv to point vi). If an error occurs at some intermediate point, the partially transformed path specification may still be reversed to correctly return an error message to the source.

Source routing simplifies routing at intermediate points by placing all responsibility for route selection at the source. When the source corresponds to a human user (perhaps accessing remote computing services from a terminal), the user establishes the initial route using whatever criteria he desires, and may update the route in response to observed performance. Unfortunately, sources communicating with many destinations may need to know the topology and performance of much of the internetwork system in order to construct successful routes. Typically less information is available to evaluate alternative routes, and changes must be infrequent (particularly for short

addressing with its essentially fixed route), so non-optimal routes will result. Although shortest path routing may be reasonably amenable to source specification, other routing criteria such as bandwidth, delay, or cost may be highly dynamic and more difficult to project from the source.

2.3 Conclusions

As a minimum for viable network interconnection, at least the following standards must be accepted by all participants:

(1) A global name space. In a hierarchical addressing system, local names within each network may remain unchanged. With source routing, globally agreed names are convenient but not necessary.

(2) Common internet routing. For all techniques this requires common address formats in internet packets and specification of routing criteria (if more than one is available). Distributed routing also involves standard routing data exchanges and routing decision algorithms.

In addition to these necessary conditions, we make the following recommendations on routing and addressing alternatives:

(1) To preserve local net independence, embedding internet packets for transmission through local nets with internet routing by Gateways is preferable to direct local net interpretation of internet packets.

(2) Hierarchical or area addressing is preferable to a single level global name space. The shorter routing tables, routing data exchanges, and local net independence gained outweigh the potential loss in routing optimality.

(3) Internet Hosts as well as Gateways must participate in internet routing (but see section 3.3 below on internet service sites).

(4) Fixed internet routing is too unresponsive to failures to be acceptable unless special failure detection and recovery mechanisms are added. Distributed routing is more robust and efficient but requires a standard universal implementation. However, distributed routing strategies are subject to catastrophic global failures when an individual node's routing process malfunctions.

(5) Source routing is most appropriate where greater source participation in route selection and non-optimal routing are acceptable in order to simplify routing at intermediate nodes or to allow more general addressing.

3. LEVEL OF INTERCONNECTION

Another basic question of network interconnection concerns the level at which networks are to be connected. A great deal of confusion is apparent in past discussions of this issue because different authors mean different things by the commonly presented alternatives, "Host" or "packet switch" level. This section attempts to clarify the concept of interconnection level by identifying three distinct issues often confused in discussions of network interconnection. Alternative approaches to each of the three isssues are explored in the following subsections with conclusions at the end of each subsection.

We maintain that at least the following three concepts represent distinct and important considerations in a coherent discussion of network interconnection level (see figure 8).

Local Net Interface Level

Here we consider that a Gateway can interface to local nets either as a packet switch (employs PS-PS protocol to communicate with local net) or as a Host (employs Host-PS protocol to communicate with local net). This is the most common meaning of interconnection level. In the packet switch case, a Gateway must behave like a normal switching and forwarding node in the local net, while in the Host case, a

FIGURE 8 INTERCONNECTION LEVEL ISSUES

LOCAL NET INTERFACE LEVEL:

- Packet Switch
- Host

LOCAL NET SERVICE LEVEL:

- Datagram
- Virtual Call
- Bulk Data Transfer
- Interactive Terminal

IMPLEMENTATION APPROACH:

- Endpoint
- Hop - by - Hop

Gateway behaves like a source/destination of local net packets (see definitions in previous section). We argue that the first alternative is neither desirable or in general even possible.

Local Net Service Level

If a Host level local net interface is chosen, a Gateway can make use of various local net service levels for transmitting packets through a local net. Levels include a simple "datagram" or best effort service, or a more reliable "virtual call" service. Other special service levels such as bulk data transfer or interactive terminal handling may also be available (see section 3.2 below for definition of service levels).

Endpoint vs. Hop-by-Hop Protocol Implementation

A desired end-end service may be implemented two ways. The Endpoint approach consists of implementing suitable control algorithms at each end of the communication path, while employing various service levels on each hop. (For example an end-end virtual call service can be provided with a SPAR (cf section II-4) type protocol at each end while using datagram services in each local net.) The Hop-by-Hop approach provides the desired end-end service without additional end-end protocol by requiring the desired service level on each hop (each local net), and joining the hops together with any necessary

translation performed in the Gateways. (For example, the same end-end virtual call service may be implemented by joining the SPAR type protocols available on each local net to each other at intermediate Gateways.) The main trade-off here is end-end controls vs. hop-by-hop controls. As we shall see below, Hop-by-Hop sacrifices some flexibility but partially avoids the need for a common internet protocol.

Treating each of these three concepts, a coherent analysis of network interconnection trade-offs becomes possible. Table 1 summarizes the classification schemes used by other authors in terms of these three issues so their results can be more easily compared.

3.1 Local Net Interface Level

Interconnecting networks through a packet switch level interface (PSLI) may appear to be the simplest strategy since the resulting "Catenet" [Pouzin73] appears to be one large network without any complicating hierarchical structure. In fact, PSLI is fraught with difficulties and has not yet been demonstrated to be feasible.

PSLI requires local nets to route internet traffic to appropriate local net Gateway/packet switches. Because each Gateway looks like a packet switch of the local net, there is no

Table 1

Network Interconnection Alternatives of Other Authors in Terms of Our Classification Scheme

	Local Net Interface Level	Local Net Service Level	Implementation Approach
Pouzin [Pouzin73]			
"Catenet"	PS	--	--
Host Level	H	VC	HH
"Super-network"	H	?	?
Davies [Davies73]			
"packet"	PS	--	E
"host"	H	VC	HH
UCL [Lloyd75a]			
"Switching node"	PS	--	--
"Parallel Host"	H	D	E
"Series Host"	H	VC	HH
BBN [Binder75]			
"packet"	PS	--	--
"host"	H	D	E
Cerf [Cerf74b]			
"host"	H	D	E

H=Host, PS=Packet switch, D=Datagram,
VC=Virtual call, E=Endpoint, HH=Hop-by-Hop,
--=does not apply, ?=unknown

way for a source Host or Gateway to specify the address of the next Gateway as a destination. Furthermore, local packet formats must be expanded to allow for internet addressing capability. Both of these requirements represent a serious breach of local net independence as discussed in section 2 above.

The main difficulty of PSLI stems from widely varying PS-PS protocols in different nets. A Gateway must somehow make the rest of the Catenet look like an extension of the local net it serves. The problem of mapping adjacent net PS-PS protocols can be considered in two parts since a PS-PS protocol involves some functions every node performs, and other functions only performed by source and destination nodes.

Universal node functions include error detection, retransmission, duplicate removal, and routing. It is reasonable to assume that a Gateway/packet switch can appropriately check and generate checksums for error detection and retransmission, and sequence numbers for duplicate removal. However, some nets may not perform node-node duplicate filtering, while others may rely on it to avoid delivering duplicates to end users. In nets that exchange routing data with adjacent nodes or a central authority, the Gateway/packet switch will have to generate appropriate routing data for the rest of the Catenet, or actually translate routing data between local nets (highly improbable given the variety of routing

practices in use). Worse problems arise with specialized node functions such as packet tracing, remote debugging, statistics gathering, call charging, etc. Gateway/packet switches will have to isolate requests for and responses from these special services to individual nets.

Even if difficulties with universal node functions could be overcome, many local nets provide extensive additional functions at source and destination packet switch nodes. EPSS requires a complicated call set-up, buffer allocation, flow control, and end-end acknowledgement scheme to be enforced by end nodes. A call between EPSS and another net would require all the appropriate fields and responses to be generated at the final destination, or at the EPSS Gateway (in which case the Gateway is behaving like an endpoint of the communication path, or a Host, not a simple packet switch). Similarly, ARPANET nodes perform storage reservation, RFNM generation, message reassembly, and message sequencing, all of which would have to be performed compatibly in the remote net. Many nets also perform accounting functions in end nodes so fees can be collected. Complying with any one local network's version of these additional functions is a formidable problem, while attempting to provide all of them in every net is clearly absurd.

Pouzin (1973, 1974a, 1974b) and Davies (1973) have been the main proponents of PSLI, citing the following advantages:

(1) Local nets will provide a simple packet transmission service without additional end node functions, making translation easier.

(2) The translation required in a Host level Gateway using Hop-by-Hop approach may be more difficult than a PS-PS protocol mapping (see section 3.3 below).

(3) The "super-network" alternative requires global agreements that will be difficult and time consuming to reach.

Point 1 appears to be the weakest since many current networks employ quite complex end-end functions (the new ARPANET type 3 messsages [Walden74] do provide a simpler transmission service). Pouzin deals with this by suggesting that extra services can be "masked out" at the Gateway/packet switch so only basic packet forwarding is maintained across the Catenet. However, this is precisely the case where a Gateway performs end node functions to cauterize local net idiosyncracies and hence is acting like a Host.

Point 2 correctly notes some of the difficulties of a Hop-by-Hop approach, but these may be reduced in an Endpoint implementation (see section 3.3). Point 3 applies primarily to providing more powerful end-end services such as virtual calls, in which case common protocols are also required with PSLI.

Interfacing a Gateway to a local net as a Host, on the other hand, has the following advantages:

(1) The Host-PS interface is typically simpler than the PS-PS interface in local nets.

(2) Local net independence is maintained because local nets do not need to know other net protocols as required with PSLI. Each local net protocol "stops" at the Gateway. All internet functions are implemented in Hosts and Gateways on top of the local net transmission services.

(3) Local nets have greater control over traffic entering from other nets since internet traffic enters from a Host.

(4) Host-PS protocol implementations typically exist for a wide range of machines, providing a headstart and a wide choice for Gateway implementation, while packet switch implementations typically exist only for a single special purpose machine.

We conclude that from both practical and theoretical viewpoints, heterogeneous network interconnection using a Host level interface is preferable to a packet switch level interface. The following sections explore other network interconnection questions assuming that Gateways use a Host level interface to local nets.

3.2 Local Net Service Level

Given that Gateways are interfaced to local nets as Hosts, several levels of service are typically available for transmitting internet packets through each local net, including datagram, virtual call, bulk data transfer, and interactive terminal services. These services may be provided by the PSN itself, or by additional protocols implemented in the Host computers. More powerful services require more complex protocols in the Host (or PSN), and more control fields in each packet transmitted (although reduced addressing may be available on fixed route connections). A brief description of each major service level follows with references for further details.

Datagram: [Walden74, Cashin74, Canada75, MacPherson75] Fixed maximum length messages are transmitted fairly reliably to the specified destination. Some messages may be lost or duplicated. Messages may also be delivered out of order. Delivery confirmation (ACK) may be available. In general no error messages are returned for lost messages or inaccessible destinations.

Virtual Call: [Carr70, Zimmerman75, Cerf74c, Cashin74, Canada75, MacPherson75, Pouzin75] This corresponds to the service level provided by Communication Control Protocols discussed in chapter II. Arbitrarily long (or at least much longer than datagram)

letters are reliably transmitted to the specified destination. Letters are never lost, damaged, duplicated, or delivered out of sequence (as long as the protocol is functioning correctly). Flow control may also be provided. Error messages are returned for inaccessible destinations.

Letters may be relatively long such as pages of a file and require fragmentation by the source CCP into smaller segments for transmission through the local net, or may be quite short as with interactive use. The important features of virtual calls are the guaranteed reliability, sequencing, and flow control.

Bulk Data Transfer: [Crocker72, EPSS75, Lloyd75a] This service is specially designed to facilitate exchange of large amounts of data (files) between end users. The virtual call service probably provides the basic transmission facility, while special file access modules are added at each end to provide for convenient manipulation of files, to chop files into records (letters) for transmission, and to reassemble records at the destination. Overhead is minimized by using long records (although limits must be imposed to avoid monopolization of net services).

Interactive Terminal Service: [Crocker72, EPSS75a, Tymes71] A large proportion of computer network traffic has been and will likely continue to be terminal access to remote service systems.

Virtual call service again provides the basis for terminal service, but because typical transmissions are short, it may prove advantageous to multiplex several sets of terminal traffic into a single physical transmission. Some code conversion and echoing control or other features may be added to the basic virtual call service [ARPANET Telnet].

Other special services such as graphics, remote job entry, or work load sharing may well prove useful and become more prevalent in the future. The concept of service levels may be pictured as a tree, with datagram (addressing and error detection) at the root, growing up through retransmission, duplicate detection, sequencing, and flow control into the virtual call level, and then branching into various special purpose services.

3.3 Endpoint vs. Hop-by-Hop Protocol Implementation

Section 3.2 has described several service levels typically available for interprocess communication on local networks. To provide convenient communication between processes on Hosts in different networks, a similar spectrum of end-end service levels is desirable. Since datagram service is likely to be even less reliable across multiple networks, and since special purpose services are likely to be built upon virtual

call service, the following discussion focuses on alternatives for providing end-end virtual call level service.

The basic choice for implementing various end-end services is between Endpoint and Hop-by-Hop or stepwise [Pouzin75] approaches:

The Endpoint approach--build the necessary control mechanisms to provide the desired service level at each end, requiring a minimum of service from the local nets in between, or

The Hop-by-Hop approach--use existing protocols on each hop (local net) to provide the service level desired, and connect the hops. Success depends on the transitivity of service: if hop A and hop B provide the service, then their connection hop AB does also. Achieving this transitivity may require a nontrivial translation between protocols.

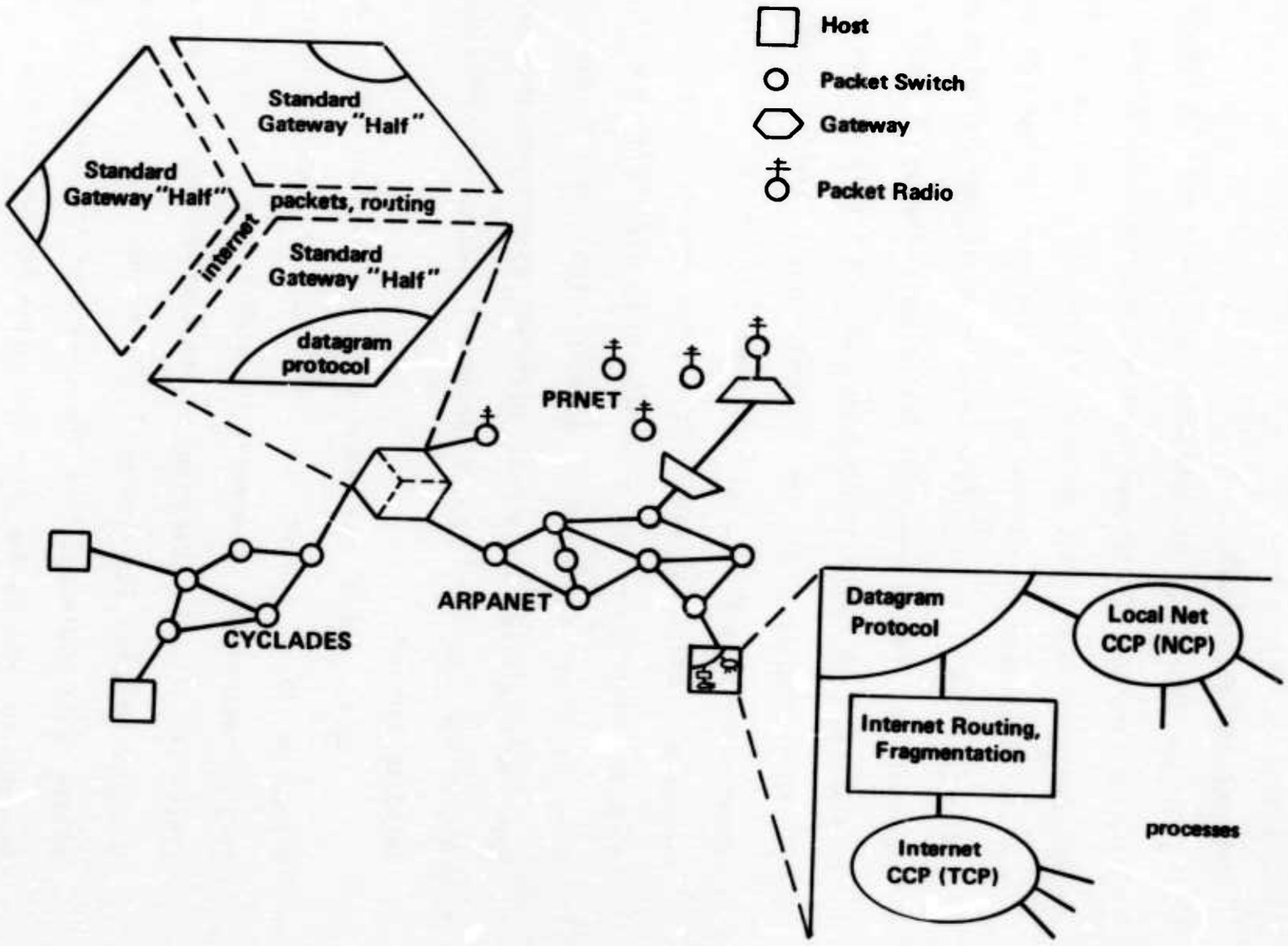
To make these alternatives more concrete, we consider the details of providing an end-end virtual call service using both approaches. Such a Hop-by-Hop implementation currently exists between ALOHANET, ARPANET, and UCL, while the following Endpoint implementation example connecting ARPANET, CYCLADES, and PRNET is still under development.

Endpoint Implementation

In the Endpoint approach, all Hosts must implement a standard internet CCP (or use local net protocols to access an internet service facility as described below). The internet CCP may be implemented in addition to an existing local net CCP (see figure 9). The internet CCPs produce internet packets which are embedded in local packets for transmission through a local net to a Gateway. A choice of local net service levels is possible for this purpose. In the ARPANET, Hosts and Gateways may communicate using the Host-Host protocol, a virtual call level connection. Alternatively, Hosts and Gateways may converse using the Host-PS protocol directly (in parallel with the NCP). Using "regular messages" this constitutes a weak virtual call type facility since the subnet performs sequencing and error correction. Using "type 3 messages" [Walden74] provides a datagram service.

Which local net service level provides the most effective total system when combined with the Endpoint CCP control mechanisms? Sequencing is undesirable in local nets since it will be performed at the destination, and increases delay in each hop (cf section II-6) as well as requiring a single exit Gateway from the local net for all packets of a connection. Hop-by-Hop error correction is more efficient than Endpoint [Metcalfe73], but for low local net error rates, the difference is small while the saving in protocol complexity from

FIGURE 9 ENDPOINT NETWORK INTERCONNECTION



eliminating retransmission on each hop may be substantial. If Hop-by-Hop retransmission is employed, Gateways must store packets until acknowledged by the next Gateway, requiring more buffer space than simply forwarding packets. Finally, some form of flow control between Gateways may be necessary, although network capacity may adequately limit traffic levels in some local nets.

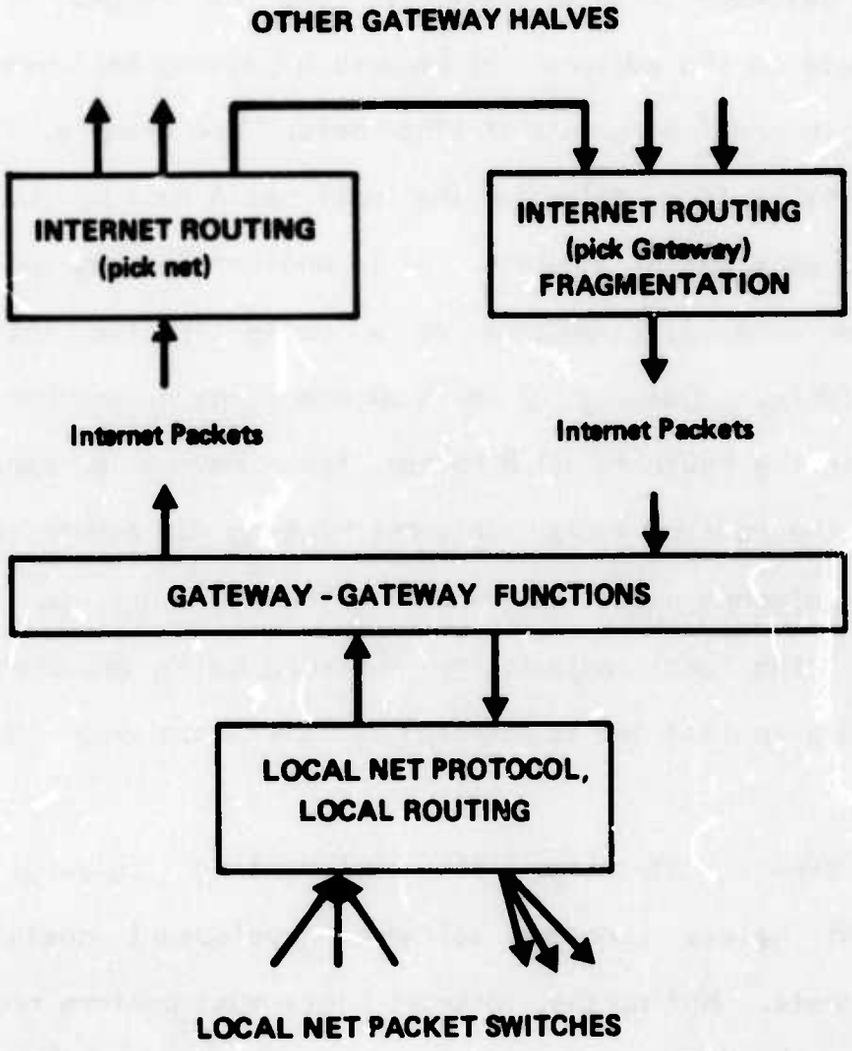
The ARPANET is an extremely low loss rate net, so there is little to be gained by local net retransmission (a service not offered by the Host-Host protocol in any case). Kleinrock, Naylor, and Opderbeck (1974) have also shown that overhead can be significantly reduced in the ARPANET by using a suitable Endpoint protocol with datagram network service rather than the network's virtual call (Host-Host) protocol. Hence the best choice in ARPANET is to use the datagram local net service, relying on the end-end internet CCP to provide additional services. The situation is similar in CYCLADES where both datagram and virtual call services are available. In PRNET where internal packet loss rates are expected to be high, local retransmission appears desirable.

Having selected a local net service level to carry internet packets, the internet packets next arrive at a Gateway. The Gateway receives internet packets via the relatively simple local net datagram protocol in most cases, and extracts the internet packets (see figure 9). The Gateway is free to send

different packets from a single connection over different internet routes since the Endpoint CCP will sequence them. Packets arriving at a Gateway need not be sequenced before forwarding. If a local route fails during the course of a connection, alternate routes may be used without causing end-end errors.

Let us call the portion of a Gateway associated with each local net a Gateway "half" (see figure 10). A Gateway half contains a local net interface (implements the necessary local net protocols) and other modules to perform internet routing, fragmentation, and any Gateway-Gateway functions (such as retransmission or flow control). The local net interface side of each Gateway half is unique to each local net, while internet packets and routing functions at the other side of each half are global. Hence a Gateway need not be a single physical device, but may consist of seperate Gateway halves for each local net, with their internet sides tied together by a simple communication line (see figure 9). Each local net could implement its Gateway half on a fully owned and controlled machine, or even as additional code on an existing Host, with different nets' Gateway halves connected by an arbitrary line control procedure for exchange of internet packets and routing data. Broadcast satellite links are particularly attractive for this purpose since they offer full connectivity and high bandwidth.

FIGURE 10 A GATEWAY "HALF"



Each Gateway half contains a routing table (RT) to perform internet and local routing as described in section 2 above. Now however, each Gateway half only maintains routes to its local net, other Gateways on its local net, and other Gateway halves (nets) to which it is connected. Routes to different Gateways in an adjacent net may be merged into a single route to the adjacent net because a Gateway half need not know the internal structure of other nets. For example, figure 11 shows the routing tables for the local net A half of Gateway G1 in the supernet of figure 4. G3 is another Gateway in local net A, and hence still appears as a route in the internet routing table. Gateway G2 is in adjacent net B, and hence is included in the route to net B rather than having a separate entry in the routing table. Internet routing now occurs in two steps: an internet packet arriving from the local net is first passed to the best adjacent net (Gateway half), and then the best Gateway in that net is selected by the receiving Gateway half.

Another advantage of implementing Gateways as independent halves concerns software development costs for internet Hosts. Noting that internet Hosts must perform routing functions identical to those in Gateways, Binder (1975) has suggested that a "logical" Gateway exists between each Host engaged in internetworking and the local net. Internet Hosts must cooperate in other Gateway-Gateway functions such as flow

FIGURE 11 ROUTING TABLES FOR A GATEWAY "HALF"

INTERNET ROUTING TABLE FOR
LOCAL Net A HALF OF GATEWAY 1

Destination Nets	Routes: A	B	G3
A	+	large	
B	∞	+	large
C	∞	3	3
D	∞	3	5

LOCAL ROUTING TABLE FOR
LOCAL Net A HALF OF GATEWAY 1

Local Destinations	PS1	PS2
A1	1	4
A2	2	3
A3	3	2
G3	3	2

+ = Local

∞ = Unreachable

control and retransmission when used, just as Gateways. Hence Gateway halves and Internet Hosts on a local net can use the same modules to perform these common functions. In the Gateway, internet packets arrive from other local nets, while in internet Hosts they are generated by an internet CCP with its additional end-end protocol functions (see figure 12).

Hop-by-Hop Implementation

Next we consider the Hop-by-Hop approach to implementing an end-end virtual call service over several networks. In this case, no internet CCP is required at source and destination Hosts. Instead each local net must provide a virtual call level service, with Gateways translating between each local net service (see figure 13). To facilitate this translation, a number of universal virtual call protocol functions can be identified:

(a) Set up call: the willingness of both parties to communicate is established, and various parameters such as letter length, window size, buffer allocation, byte size, echo mode, and abbreviated addressing are agreed on.

(b) Terminate call: the connection is broken either immediately or after any letters in progress have been delivered.

(c) Send a letter.

(d) Receive a letter.

FIGURE 12 GATEWAY "HALF" IN AN INTERNET HOST

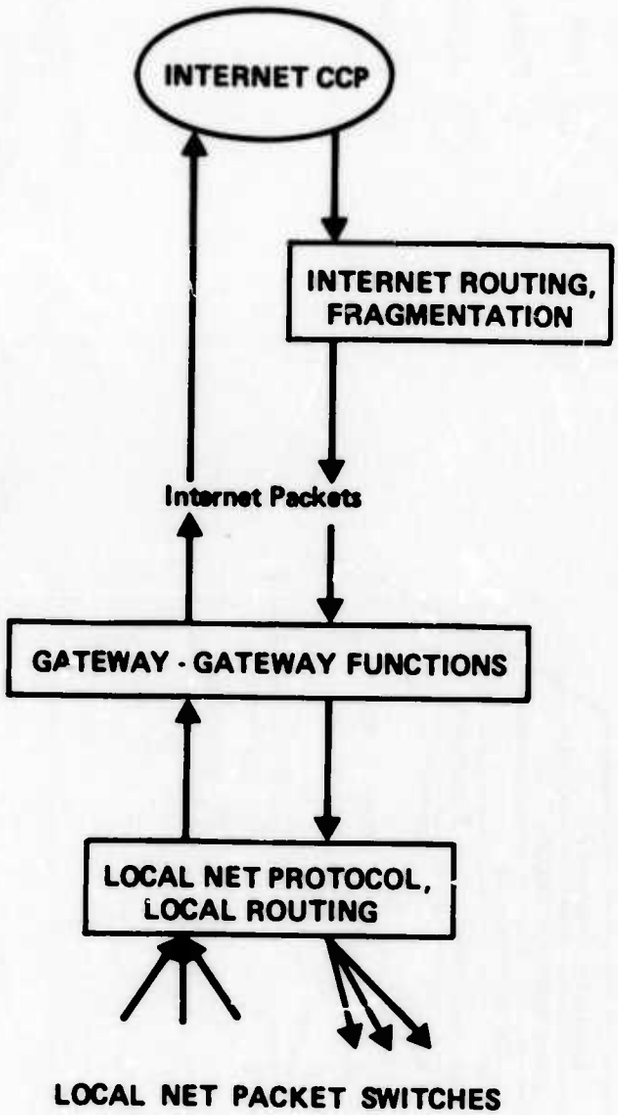
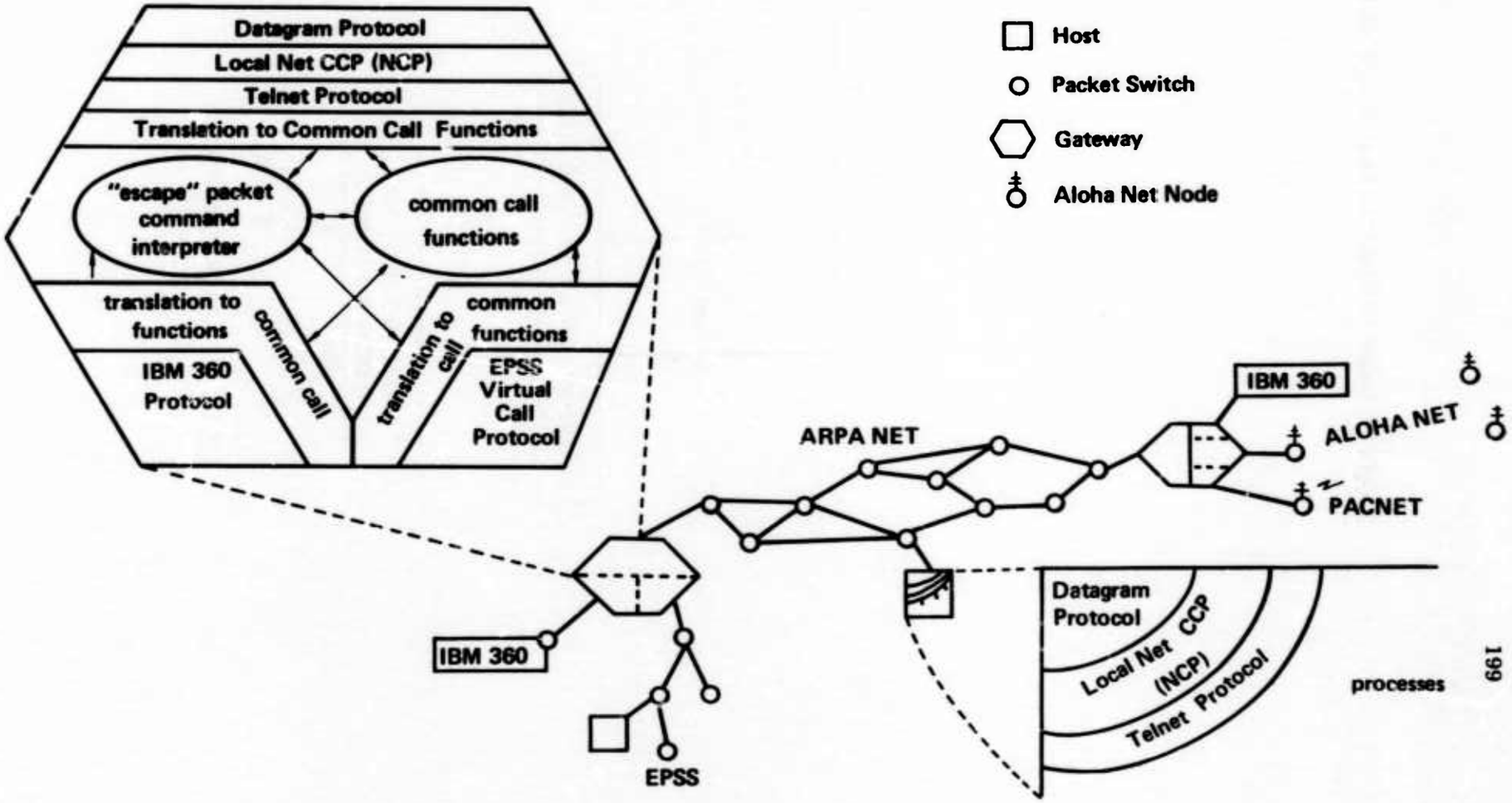


FIGURE 13 HOP - BY - HOP NETWORK INTERCONNECTION



(e) Signal an interrupt, reset, or attention.

(f) Obtain status information on letters pending or call parameter values.

Consideration of this list indicates that translation may be difficult. In some cases similar but incompatible services are provided by different local net protocols, such as letter based, byte based, or line based flow control. The difficulties of interfacing different flow control mechanisms are frequently not appreciated [UKP074, Stokes75]. Other services such as status, echo control, or interrupt may not be provided at all by some local nets. In this case the Gateway must simulate compliance locally without being able to obtain the service at the ultimate destination. In general this reduces internet services to the subset of services offered by all local nets, or requires the end user to be aware of what services he is "really" getting depending on the particular local nets traversed.

An alternative to "masking" services without counterparts in subsequent local nets as above, is to add the missing services with extra modules outside the local net protocol. This may be suitable between Gateways of a local net, but internet Hosts must also add the modules to existing protocols, resulting in a modified interface for internet communication after all. Although the addition will be smaller

than a full Endpoint CCP, there is a quantum jump in inconvenience when any change to the user interface is necessary.

To provide a concrete example of Hop-by-Hop implementation, consider interconnecting ALOHANET, ARPANET, and UCL where this approach has been used [Binder74, Higginson75]. A major difference of these implementations from the Endpoint approach is that a user is required to explicitly set up the call on each hop. For example, an ALOHANET user may first connect to his local Gateway using ALOHANET protocol, then connect to UCL using ARPANET Telnet protocol, and then command the UCL Gateway to connect him to the IBM360. Once the connection is established, data is forwarded automatically by each Gateway.

Both ALOHANET and UCL Gateways have developed a similar set of commands/functions to those listed above, to facilitate interconnection of the several nets or computers connected to each Gateway. Some special functions (e.g. Interrupt) are automatically translated by both Gateways. Other functions are only implemented in some nets (e.g. echo control in ARPANET) or some Gateways (e.g. Status and Operator functions at UCL) and cannot be translated or automatically forwarded. These functions must be explicitly requested by using "escape" characters interpreted by one of the Gateways (rather than passed on as data). As noted above, the end user also sets up

each hop of the connection by using command packets to communicate with each Gateway in turn.

It may be possible to automate connection establishment by having Gateways forward a standard call set-up packet, but it will still be necessary for a user to specify an entire path or at least the final internet destination in some nonstandard (to the local protocol) fashion. Unfortunately, many special Telnet services have no direct counterpart in ALOHANET or UCL nets, so automatic translation of such services is problematic. Similar difficulties must be expected with the implementation of other special purpose protocols such as bulk data transfer where local net differences are likely to be greater (for example see [Stokes75]).

Another difference in the Hop-by-Hop approach (for virtual call service) is that a single internet path (of Gateways) must be used between source and destination. When one hop fails or malfunctions, end-end service is affected since there are no corrective end-end control mechanisms.

As a final difference, each new Gateway or network added with the Hop-by-Hop approach presents a unique translation problem between the protocols involved. Acceptance of virtual call protocol standards may simplify this problem somewhat. On the plus side, only bilateral agreement between connecting networks is required for translation of local protocols.

Comparison of Approaches

We can now make a number of comparisons between Endpoint and Hop-by-Hop interconnection strategies described above. UCL has been the primary advocate of a Hop-by-Hop approach, citing the following advantages [Lloyd75a, Lloyd75b, Higginson75]:

(1) Only bilateral agreement is required, allowing immediate development and implementation, while Endpoint requires difficult and time-consuming multilateral agreements.

(2) Existing local net protocols are employed to full advantage and no new internet CCP is required, reducing software development and user accomodation to a minimum.

Point 1 is well taken and argues strongly in favor of Hop-by-Hop as a quickly available interim implementation. Point 2 requires closer scrutiny considering both internet Hosts and Gateways. For Gateways, point 2 seems to be incorrect. As shown above (for end-end virtual call service), the Hop-by-Hop Gateway requires a local net virtual call protocol, plus a unique (to each network pair) translation between local net protocols, or mapping into some set of universal call functions. The Endpoint Gateway requires smaller, simpler, local net datagram protocol (with possible additions of some locally desirable extra functions), and no translation. Endpoint Gateway "halves" for a given local net are all identical,

avoiding the unique translation effort required in the Hop-by-Hop approach. Both types of Gateway require internet routing, fragmentation, accounting, and other auxiliary functions discussed in the next section. The Endpoint Gateway does not contain "a complete internet Host for each attached network" as stated in [Lloyd75], but only the internet routing and related functions common to Gateways and internet Hosts. In particular, there is no end-end internet CCP in the Endpoint Gateway (unless the Gateway is also an internet Host). In summary, Endpoint Gateways appear to require less total software, and less new software development than Hop-by-Hop Gateways.

Within Hosts, point 2 appears more justified since the Hop-by-Hop approach does maintain existing local net protocols. However, significant user intervention has been necessary to complement local protocols for purposes of internetworking as discussed above.

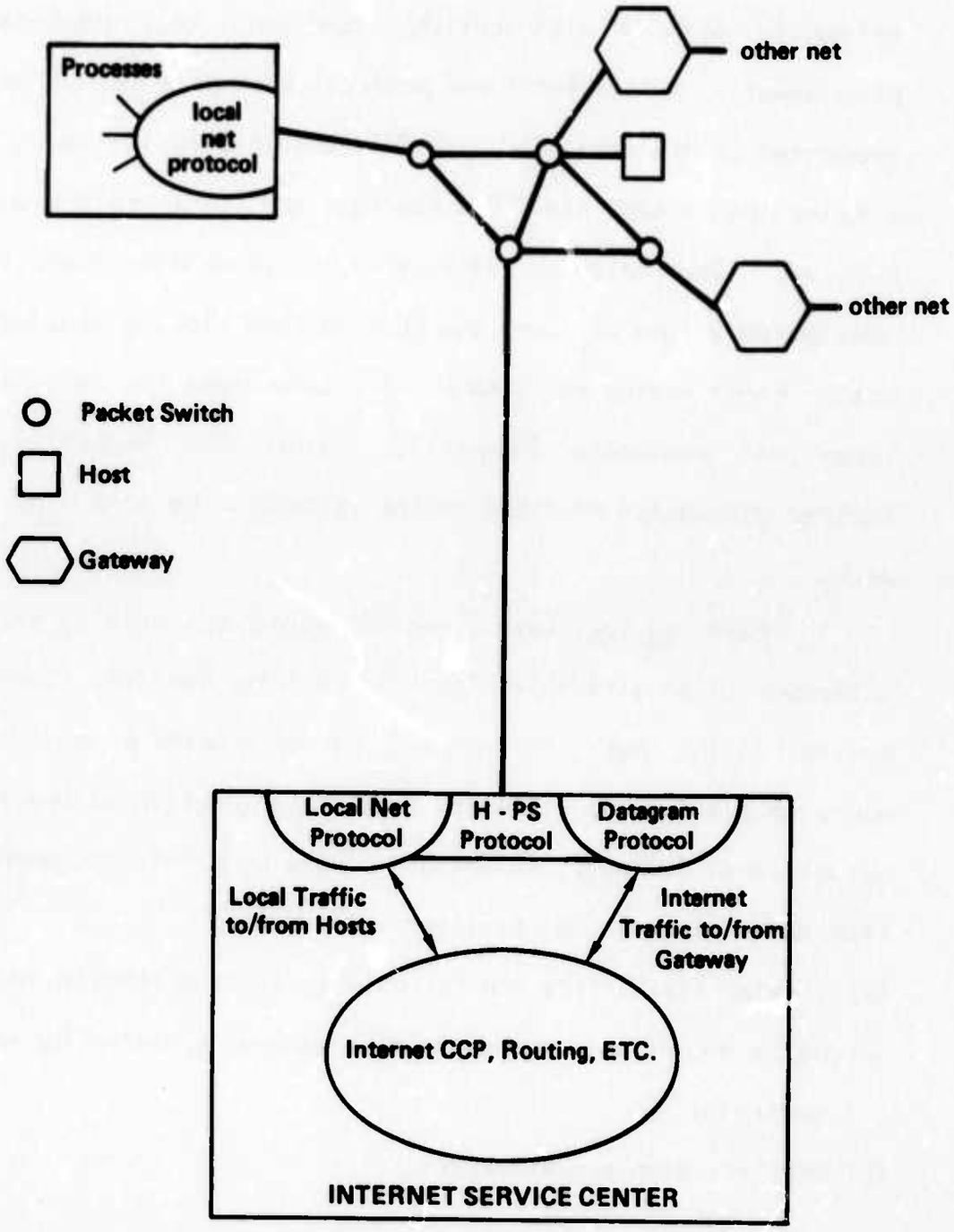
Several considerations reduce the difficulty of Endpoint internet Host implementations requiring an internet CCP. In new networks, internet standards can be implemented from the start. In existing networks, the development of high level language implementations of standard Gateway functions may reduce local net development efforts to local net interface portions (which already exist for local Hosts). The internet routing and fragmentation portion of Gateway functions are identical in all

Gateways and internet Hosts, and may be transportable from other implementations.

From the user's point of view, an Internet CCP may provide an interface that is similar to existing local net protocols so changeover should be facilitated. Where adoption of internet CCP at a particular Host site is not possible (inadequate facilities, fixed system, effort not justified by expected use), connection through existing local protocols to an internet service site may be used (see figure 14). The internet server might provide all CCP and Gateway functions, with the local net serving essentially as an access line between processes and (remote) CCP. The local net access line may degrade performance of the CCP which is no longer fully end-end. This strategy shares some of the disadvantages of a Hop-by-Hop approach, but allows Hosts with limited facilities (intelligent terminals, packet radio units) to make use of internetwork facilities.

As a more robust compromise, individual Hosts may provide an internet CCP but not the additional Gateway functions (primarily internet routing). In this case the Hosts transmit locally generated internet packets to a simpler internet service facility which provides Gateway functions (a kind of central office for routing and supernet control, or perhaps a normal Gateway). Some sacrifice in routing optimality and robustness compared to a complete internet Host may result.

FIGURE 14 INTERNET SERVICE CENTER



Another technique for reducing the burden of internetwork communication facilities on Host resources is to place the CCP in a front-end processor. This moves most communication functions to the front end, leaving the Host primarily with a distribution function (of packets to processes). A Host-front end protocol must also be defined and supported in the Host [Padlipsky74], but this is typically much simpler than a complete CCP since Host and front end are closely coupled. The internet service site described above may be considered a type of front end that is less closely coupled to a Host. Front ending may also provide advantages in implementing local net protocols [Newport72, Feinroth73, Benoit74], but further discussion of front ending is beyond the scope of this work.

Cerf et al have been the main proponents of Endpoint interconnection strategies [Cerf73, Cerf74a, Cerf74b, Cerf74c]. Davies (1973) has also argued the advantages of an Endpoint approach although he discussed them in conjunction with a packet switch level Gateway. Advantages of the Endpoint implementation (for an end-end virtual call service) include:

(1) Greater flexibility and reliability since alternate Internet paths may be used and path failures are recovered by end-end controls.

(2) Smaller, simpler Gateways.

(3) A single Gateway function module may be used at all Gateways and internet Hosts on a local net.

(4) Addition of new nets or supernet topology changes are easy because the interface between Gateway "halves" is universal. The internet side of Gateway halves for any local nets can be connected without special modifications.

(5) Fragmentation is simpler and more flexible (see next section).

(6) Acknowledgements and control functions are truly end-end.

(7) The internet CCP provides a uniform user interface that "really" provides specified virtual call services. Hop-by-Hop must mask (fake) some services, augment existing user interfaces, or require explicit user intervention at individual hops.

(8) The same Gateway implementation and local net services can be used to provide different end-end internet services by providing different protocols at the ends in the internet CCP. For example bulk data transfer or a different virtual call service could be implemented in internet Hosts with no change to Gateways. In the Hop-by-Hop approach, different local net protocols and a new translation between them in the Gateway would be required for each end-end service.

In conclusion, Endpoint and Hop-by-Hop interconnection strategies appear best suited for different situations. Both

merit further experimentation to verify projected advantages and difficulties. (Such experimental plans are outlined in [Binder75], [Lloyd75a], and [Cerf74b].) A Hop-by-Hop approach appears most suitable for backward compatability (see [Burchfiel75]), minimum effort, or immediate need applications, while an Endpoint approach offers greater robustness and generality but requires substantial standards and new software development. Development costs should be reduced by the universal applicability of internetworking modules in the Endpoint approach.

4. ADDITIONAL GATEWAY FUNCTIONS

The central questions of internet routing, addressing, and Gateway implementation alternatives have been discussed in the previous two sections. This section considers a number of additional Gateway functions important to network interconnection.

Fragmentation

A great deal of discussion has surrounded the issue of fragmentation. One way to avoid the difficulties of differing local net packet size limits is to establish a standard minimum maximum packet size for all nets, and never use packets larger than the standard for internetwork communication.

Such a standard appears ill-advised for several reasons. Since network performance depends heavily on packet size, networks designed for different purposes will have good reasons for different local packet sizes, and agreement on a standard length will be difficult. Too small a standard results in high overhead for internetwork packets with typically long headers (lengths over 250 bits have been proposed), while a long standard will be difficult for some nets. Nets with smaller limits might be able to fragment packets at entry and reassemble them at exit to comply with the standard. However, this sacrifices the flexibility of alternate routing which becomes

impossible when all fragments must exit through the same Gateway in order to be reassembled. Finally, rapidly changing technology including high bandwidth digital circuits and broadcast transmission will certainly dictate revision of optimal packet sizes.

In general, a source CCP is obliged to fragment large letters for transmission into its local net. With Hop-by-Hop network interconnection, a Gateway must translate between fragmentation schemes in each local net, possibly further fragmenting oversize fragments or letters. Where such translation is impossible, the Gateway may have to reassemble each letter and refragment it compatibly with the next local net protocol.

Endpoint network interconnection allows a simpler and more flexible fragmentation strategy. Individual nets may use arbitrary packet sizes, while Gateways fragment oversize internet packets at entrance to a local net, and reassembly of fragments occurs at the destination CCP. The internet header includes fields to control reassembly of letters at the destination CCP. It is quite straightforward to allow a Gateway to (further) fragment internet packets by using the same fields as the source CCP. The entire internet header is copied for each fragment created, with alterations only to indicate the new text length and order of each new fragment. Fragments can then be forwarded independently to the destination where they are reassembled exactly as if generated by the source CCP.

Cerf (1973, 1974c) has proposed a fragmentation control scheme based on byte sequencing of data on a connection, while McKenzie (1974) has proposed a similar scheme based on larger units of data. Smaller units allow smaller fragments of a letter to be transmitted and finer grained flow control, but require longer packet identifier and acknowledgement fields [Day75a]. Longer fields are required to ensure reliability because sequence numbers must not be reused for a maximum packet lifetime (cf section II-4), and small units consume sequence numbers at a faster rate for a given throughput. LeMoli (1975) has suggested a hierarchical fragment notation which would be carried in a separate field of the internet header. This fragmentation scheme would operate independently of other aspects of the end-end protocol in contrast to the Cerf proposal where the sequence number field serves both reliability (cf section II-4) and fragmentation purposes. Any of these schemes appear suitable for Endpoint interconnection strategies where the Gateways and internet Hosts all share a uniform fragmentation mechanism.

Although adoption of such a scheme allows arbitrary local net packet sizes, it complicates selection of optimal packet sizes for internet communication. Presumably small packets (from interactive traffic) will traverse all nets on a one-to-one basis with no complications. High throughput applications, on the other hand, tend to use large packet sizes

to reduce overhead. In this case, passing through even a single "small packet" net may cause degradation, since once packets are fragmented in the small packet net, they are not reassembled. All the fragments must be carried through subsequent nets which might have accepted the original packets more efficiently, or at lower cost. In such cases a user may wish to forego the added robustness of independent fragment propagation in favor of local net fragmentation/reassembly.

Cost minimization may prove a very slippery problem as shown by the following example. Fees in a PSN may be charged per packet (regardless of length), or per bit. Ignoring other factors such as distance or service level, cost may be measured as packets per bit (of data) or total bits per bit (of data). Assume a user in net A wishes to send a large amount of data to another net A user. Suppose net A allows 4000 bit packets while net B allows only 1000. The internet packet header is 200 bits long. Strategy 1 minimizing both cost measures in net A is to send maximum length packets with 3800 bits of data.

Now suppose a net A user wants to communicate with a net B user. Strategy 2 attempts to optimize transmission of fragments through net B by sending 3200 bits of data in each net A packet, allowing fragmentation into four full packets in net B. Table 2 shows the packet and bit costs resulting from the two net A packet sizes, assuming charges are the same in both nets. Strategy 2 reduces the packet cost as expected, but

Table 2

Results of Internet Fragmentation for Two Packet Size Strategies

data/packet in net A	packets in net A	total bits in net A	packets in net B	total bits in net B	TOTAL COST bits/bit	TOTAL COST packets/bit
3800	1	4000	5	4800	2.32	1.58*
3200	1	3400	4	4000	2.38	1.53*

Maximum packet size
Net A: 4000
Net B: 1000

Internet Header Size: 200

* $x10^{-3}$

increases the bit cost (the overhead reduction in net B is offset by the increase in net A).

In general, neither the route followed nor the fragment sizes generated on a communication path are constant or even known. Hence such attempts at cost or performance optimization become quite difficult. Fortunately, cost differences are small if a reasonably large maximum packet size is available in all nets (a packet size ten times the internet header size has an overhead of 0.11 for full packets).

Accounting

As indicated above, local net fees may be based on several factors including number of packets, number of bits, distance, connect time, and service level (reliability, bandwidth, delay). Presumably each local net has effective techniques for recording charges and collecting fees from local Hosts. With network interconnection, the Gateways present a new source of traffic that must be charged. Packet switch level Gateway interfacing presents great difficulties here since local nets do not normally charge traffic from adjacent packet switches.

Local nets are normally equipped to charge traffic from Hosts, making Host level Gateway interfacing preferable for accounting purposes. Nevertheless, a local net is likely to

charge on the basis of total traffic from the Gateway Host without distinguishing the sources of such traffic. Hence it is up to the Gateway "half" for each local net to monitor traffic levels from each source of interest. For outgoing (from the local net) traffic, the Gateway may wish to separately account for traffic from each local Host (including other Gateways). For incoming traffic (from other nets), discrimination on the basis of net may be adequate. Outgoing traffic from one net's Gateway half becomes incoming traffic to the Gateway halves of connected networks, so each local net authority will be in a position to verify charges from adjacent nets.

In establishing internetwork charges, other existing internetwork communication systems (Post Office, telephone, telegraph) provide well developed examples. Presumably each local net authority will collect both local and internet fees from local users, exchanging accumulated internet charges with other nets periodically. If charges depend on the internet route (number or identity of nets traversed), users may desire the option to control routing, or at least to specify "minimum cost" routing. This may significantly complicate the routing algorithms.

Status Monitoring and Reporting

In secton 2 on routing, the "reachability" of local Hosts was mentioned as an important part of the routing data base. Each Gateway half is responsible for knowing the reachability of the Hosts (CCPs) on its local net. This is particularly desirable for virtual call protocols which retransmit to achieve reliable communication. A "destination inaccessible" error message must be returned to the source CCP in order to quench useless retransmissions, and even more importantly, as information for the user. Useful subtypes of such a message might specify the level of failure (Net, Gateway, Host, CCP, Process, Port), and the reason (nonexistent, dead, busy) [Cerf74b].

Local nets often maintain Host accessibility data as part of their local routing procedures. In local nets where internet packets are embedded, the local net may generate an error message and return it to the Gateway which was the local net source of the packet in error. The local net cannot signal the internet source directly since the local net is not aware of the Internet header and Internet signalling conventions. On the basis of a local net error message, the Gateway can mark a local Host as inaccessible, and return Internet error messages for subsequent packets destined for the inaccessible destination. Such remote signalling is more difficult with Hop-by-Hop network interconnection where control signals must be translated between

local net protocols as discussed in section 3. The Gateway may also determine Host accessibility on its own by periodically exchanging status packets with local Hosts or monitoring internet traffic to local Hosts.

Other status information such as loads, traffic levels, expected delays, charges, availability schedules, current users, current line conditions, etc. may be maintained by local nets, Hosts, or Gateways, and made available by internet status inquiries. Some local net status information is directly usable at the internet level (for example accessibility data as above), while other local net data may not be useful (for example transmission error messages when no Hop-by-Hop retransmission is employed). Special services such as tracing, echoing, or discard may also be defined at Gateways and internet Hosts.

Appendix A

CONNECTION ESTABLISHMENT PROOFS

In this appendix we present a proof that the protocol initialization mechanisms discussed in chapter II correctly establish a connection for reliable communication between the processes served by the protocol. The proof extends the methods of Gilbert and Chandler (1972) to distributed systems with "thin wire" interprocess communication [Metcalfe72]. Gilbert and Chandler defined the "composite state" of a system as the state of each process in the system plus the value of shared variables in a common memory. Since communication protocols interact by exchanging messages rather than shared variables, this model is not directly applicable.

We define a similar composite state of a Communication Control Protocol (CCP) as the state of the protocol process on each side of the connection, plus any "relevant" packets in the transmission medium between them. The two protocol processes are modeled as (identical) state machines operating independently except for the explicit exchange of packets. Synchronization of the two processes is achieved when one CCP waits for a particular type of packet from the other CCP. Transitions from one composite state to another are derived from the state transitions of the individual protocol machines (Gilbert and Chandler's "partial rules").

Another major difficulty in applying state models to communication protocols is the large number of states that must be considered for a protocol of even modest complexity (cf section 1.2 of chapter II). We overcome this problem by severely limiting the number of packets which must be considered part of the composite state. In a straightforward approach, every packet from the time the composite system was "created" would have to be represented. By considering the protocol's use of sequence numbers and control packets, all packets in the transmission medium can be classified as either "current" or "old" packets. Since we are primarily interested in worst case analysis, we assume that any old packet may be delivered to a protocol machine at any time (limited by maximum packet lifetime). Hence only current packets must be explicitly represented as part of the composite state.

Our composite state model helps answer several interesting questions about the reliability of connection establishment, such as:

(1) Forbidden states or state sequences: Does the protocol ever reach undesirable states (e.g. accepting old duplicate data)?

(2) Deadlock: Does the protocol reach a state where each side is waiting for the other and neither can proceed?

(3) Halting: Does the protocol eventually establish a connection once it sets out to do so?

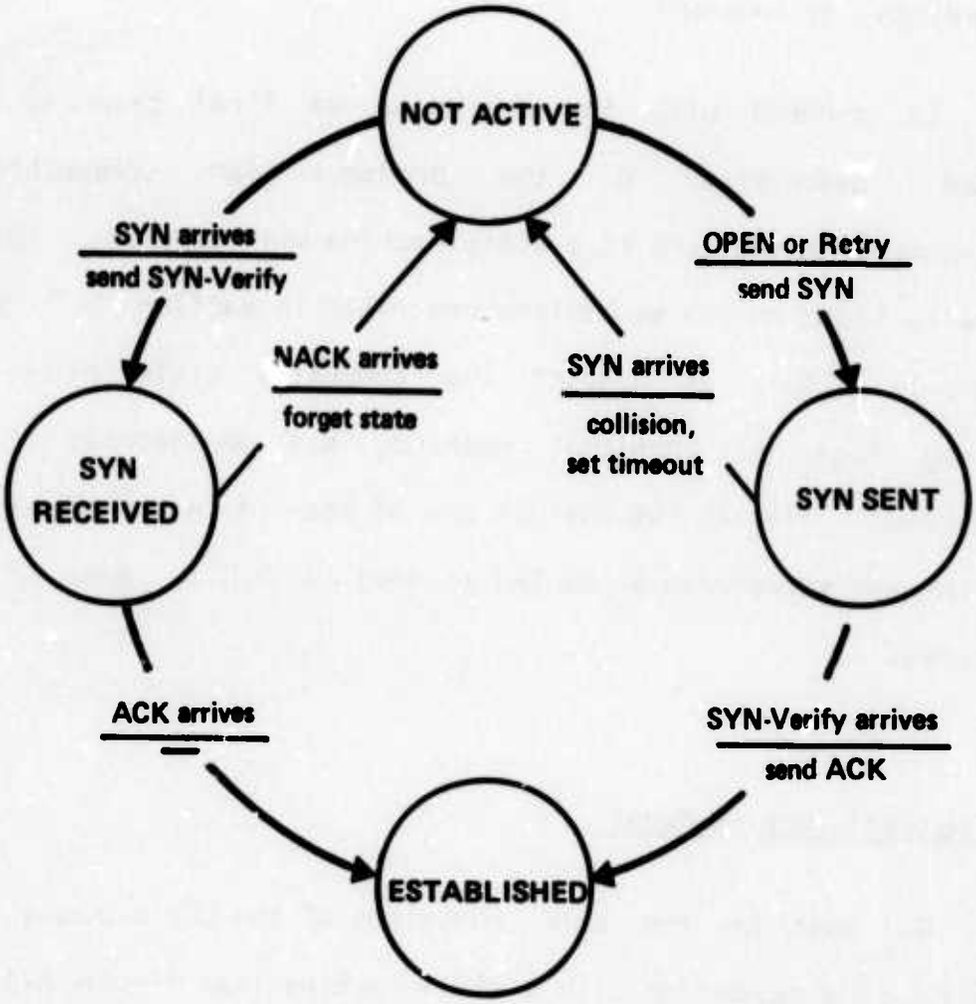
(4) Failure consequences: When one side of the protocol fails (for example due to a Host crash), does the protocol deadlock, or recover?

To proceed with the analysis, we first provide a detailed description of the protocol for connection establishment in the form of a state machine with context. This machine is based on the mechanisms described in section 5.2 of chapter II. Next we present the composite state diagram resulting from this protocol machine, and demonstrate its reliability. Finally we analyze one of the simpler connection establishment procedures presented in section 5.2 to show its weaknesses.

A.1 Protocol Machine Model

We describe the main functions of the CCP process on each side of a connection with a state machine (see figure A-1). A state machine model includes states (represented as circles in figure A-1), transitions from one state to another (represented as directed arcs), events which cause the transitions (written above horizontal bars in the figure), and actions associated with each transition (written below events).

FIGURE A - 1 "3 WAY HANDSHAKE" PROTOCOL MACHINE



Four major states are necessary to model the 3 way handshake connection establishment mechanism (cf section II-5.2):

Not Active (NA): The protocol is not initialized and a minimum of information about the connection is maintained

SYN Received (SR): A SYN control packet has been received, and a SYN-Verify returned to the remote protocol, but the final ACK (third part of the 3 way handshake) has not been received

SYN Sent (SS): In response to an OPEN command from a process, a SYN control packet has been sent to the remote protocol machine, but no SYN-Verify response (second part of the 3 way handshake) has been received

Established (ES): The 3 way handshake is complete and the protocol is initialized for reliable data communication

In addition to the major state, each CCP maintains a context of additional information about a connection, including sequence numbers and pending packets as described in section II-4, parameter values such as flow control window size, quit time, or retransmission timeout, and internal timers. Use of this context information in the protocol machine model reduces the number of states required to represent protocol operation and further simplifies the composite state analysis.

Events activating the protocol machine include packets arriving from the transmission medium, commands from the local process, and internal timeouts. The events relevant to connection establishment are:

Packet Arrivals (without error)

SYN: a SYN control packet

SYN-Ver: a SYN-Verify control packet

Data: a Data packet

ACK: an Acknowledgement packet

NACK: a negative acknowledgement (of a SYN or SYN-Verify)

Reject: a Reject control packet

Process Commands

OPEN: open a connection

Internal Timeouts

Retrans: a pending packet requires retransmission

Quit: the Quit time for a pending packet has expired

Retry: the collision retry timeout has expired

All of these events have been described in sections 4 and 5 of chapter II except the Reject control packet which has been added to allow a process to refuse attempts to establish a connection.

The operation of the protocol is represented by the transitions and their associated actions. The current state, the event occurring, and the context together determine the action to be taken and the next state. Figure A-1 shows the transitions normally occurring during connection establishment. For completeness, the occurrence of all events in each state must be considered. Since this would result in an overly

complex drawing, tables A-1 to A-4 provide a complete description of the transitions from each of the four states. Each transition is named for later reference with the two letter code of the current state followed by an integer. Then the event causing the transition is given, followed by the next state, the action taken, and any relevant context tests. Events causing the same action and next state are grouped under a single transition name.

The set of operations (detect event, take appropriate action, move to new state) are assumed to be atomic or uninterruptable so that no confusion can result from nearly simultaneous events. In a real implementation, this may require some sort of lock facility to defer later events while the operations triggered by an earlier event are completed.

The transitions are based on the description of connection establishment mechanisms in section II-5, particularly the 3 way handshake. The simple collision recovery technique is used because it allows a simpler model. Retransmission and Quit times apply to control packets (SYN and SYN-Verify) as described for data packets in section II-4.

Reject packets are returned in response to a SYN (transition NA2) when the receiver is unwilling to establish a connection. When the Reject reaches the initiating CCP, the CCP can cease retransmitting SYN and notify its local process of the reason for rejection (transition SS7). The transmission medium

Table A-i

3-Way-Handshake Protocol State Transitions
from Not Active State

Name	Event	Next State	Action and Context
NA1	SYN	SR	Send SYN-Verify referencing SYN. Remember seq. no. of SYN.
NA2	SYN	slf	Send Reject referencing SYN if busy.
NA3	SYN-Ver	slf	Discard, send NACK referencing SYN-Verify.
NA4	Data, ACK, NACK, Reject	slf	Discard as old.
NA5	OPEN, Retry	SS	Pick ISN and send SYN.
NA6	Quit, Retrans	slf	Does not occur since no packets pending.

slf = self, SR = SYN Received, SS = SYN Sent

Table A-2

3-Way-Handshake Protocol State Transitions
from SYN Received State

Name	Event	Next State	Action and Context
SR1	SYN	slf	Send ACK if SYN is for current incarnation.
SR2	SYN	slf	Send NACK if SYN is not for current incarnation.
SR3	SYN-Ver	slf	Send NACK referencing SYN-Verify.
SR4	Data	slf	Discard as out-of-order (or hold).
SR5	ACK	ES	If ACK refers to pending SYN-Verify. Third part of 3 way handshake.
SR6	NACK	NA	If NACK refers to pending SYN-Verify. SYN previously received was an old duplicate.
SR7	ACK, NACK, Reject	slf	Ignore if does not refer to pending SYN-Verify.
SR8	OPEN	slf	Ignore since already in progress.
SR9	Retrans	slf	Retransmit pending SYN-Verify.
SR10	Quit	slf	Notify local process.
SR11	Retry	slf	Ignore since other side has already started.

slf = self, ES = Established, NA = Not Active

Table A-3

3-Way-Handshake Protocol State Transitions
from SYN Sent State

Name	Event	Next State	Action and Context
SS1	SYN	NA	Set collision retry timer.
SS2	SYN-Ver	ES	Send ACK if SYN-Verify refers to pending SYN. Second part of 3 way hardshake.
SS3	SYN-Ver	slf	Send NACK if SYN-Verify does not refer to pending SYN.
SS4	Data	slf	Discard as out-of-order (or hold).
SS5	ACK	slf	Ignore old ACK.
SS6	NACK	slf	Ignore and retry. Other side has received an old SYN.
SS7	Reject	NA	Notify local process if Reject refers to pending SYN.
SS8	Reject	slf	Ignore if Reject does not refer to pending SYN.
SS9	OPEN, Retry	slf	Ignore since already in progress.
SS10	Retrans	slf	Retransmit pending SYN packet.
SS11	Quit	slf	Notify local process.

slf = self, NA = Not Active, ES = Established

Table A-4

3-Way-Handshake Protocol State Transitions
from Established State

Name	Event	Next State	Action and Context
ES1	SYN	slf	Send NACK referencing SYN.
ES2	SYN-Ver	slf	Send ACK if for current incarnation.
ES3	SYN-Ver	slf	Send NACK if for previous incarnation.
ES4	Data, ACK, Quit, Retrans	slf	Handle data communication as described in section II-4 for SPAR protocol.
ES5	NACK, Reject	slf	Ignore old duplicates.
ES6	Open, Retry	slf	Ignore since accomplished.

slf = self

may also return Reject packets if a destination is unreachable for various reasons (see chapter IV). Both of these applications are conveniences that provide processes with more information than simply timing out on their quit time.

As mentioned at the beginning of this appendix, we are particularly interested in analyzing protocols under worst case conditions. Therefore we assume that duplicate packets from previous incarnations of a connection may be held in the transmission medium and emerge during or after establishment of the current incarnation. This means that all packet arrival events can occur in every state, even though some packet types would not appear if the transmission medium delivered packets in order. This assumption precludes analysis techniques which depend on in-order packet transmission (cf section II-1.2).

Sequence numbers are used throughout the protocol to uniquely identify packets. Section II-4 and II-5 have presented constraints for assigning sequence numbers to packets, and demonstrated the serious errors that can occur when these constraints are violated. As a basis for the correctness proof in this appendix, we assume that window size, transmission rate, and initial sequence number selection constraints are obeyed. This guarantees that on any connection there exists at most one (different) packet with a particular sequence number.

A.2 Composite State Model

Construction of a composite state diagram from the protocol machine in section A.1 is primarily a mechanical process. Each protocol machine transition applicable to either of the processes in the composite state becomes a composite state transition. Any packets generated as part of the action of the transition are added to the current packets in the composite state. Packets are removed from the composite state when they are no longer "current." A packet is current if either:

(1) The packet is pending (waiting for retransmission) at the sender. Normally this condition holds until the sender receives some form of acknowledgement. For packets which are not retransmitted (ACK, NACK, Reject) it does not hold.

(2) The packet refers to a current packet traveling in the opposite direction (e.g. ACK, NACK, or Reject of another packet). When the opposite packet is no longer current, both packets are removed from the composite state.

Limiting packets explicitly considered part of the composite state to these current packets is possible because we assume that any "old" packet (including those removed from the composite state) may arrive at any time. If the protocol performs correctly under this worst case assumption, it will

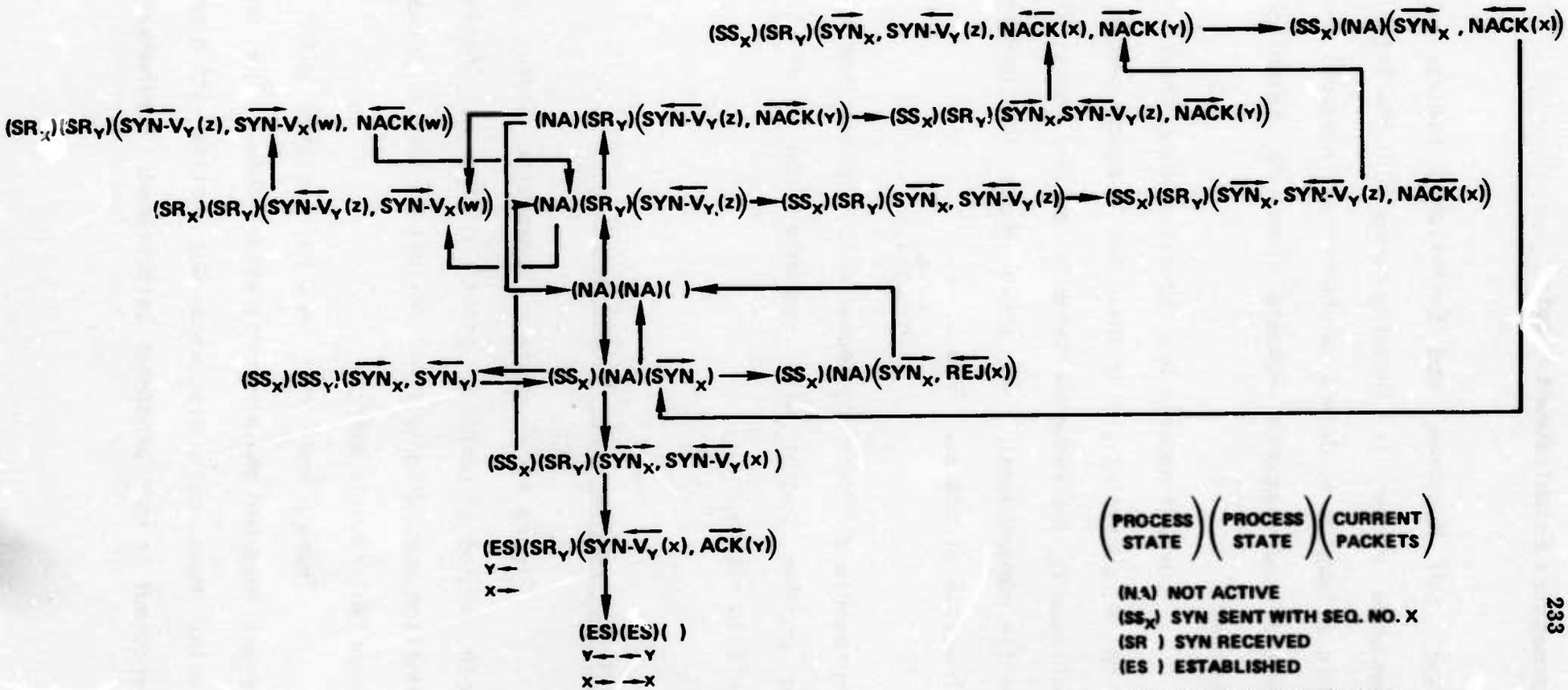
also perform correctly under conditions where only a subset of all possible old packets may arrive late and out-of-order.

Once a packet is generated by a protocol transition, it remains in the composite state until it is no longer current, despite the assumption that the transmission medium can lose or damage packets. Since every current packet is either being retransmitted, or is a response to a packet being retransmitted, we can assume that current packets are always available to cause transitions. There are no time limits on transitions occurring in the model. In reality, packets may temporarily "disappear" from the composite state if lost or damaged, but will always reappear due to retransmissions.

Figure A-2 shows the composite state diagram for the connection establishment protocol defined in section A.1. Symmetric states (identical except for switching process identities and packet directions) have been eliminated to simplify the figure. Transmissions to the same state such as retransmissions are not shown. Composite transitions resulting from simultaneous transitions of both protocol machines are perfectly legal, but are shown as sequential individual transitions to reduce the number of arrows.

Each composite state is represented by a pair of process states and a list of current packets. Some context is represented along with the basic state of each process. This consists of the sequence number for outgoing packets in the SYN

FIGURE A - 2 COMPOSITE STATE DIAGRAM FOR "3 WAY HANDSHAKE" PROTOCOL



Sent, SYN Received, and Established states, and also the sequence number for incoming packets in the Established state. This allows us to determine whether the protocol has correctly initialized sequence numbers when the Established state is reached.

Current packets are represented by their event names, with a subscript giving their own sequence number if relevant, followed by the sequence number of another packet they may refer to (in parentheses). An arrow above the packet shows its direction of travel. Thus

$$\overrightarrow{\text{SYN-V}}_{y}(x)$$

represents a SYN-Verify packet with sequence number y, referring to another packet with sequence number x, and traveling from left to right.

A.3 Correctness Under Normal Operation

Figure A-2 presents all composite states reachable if both protocol machines start in the Not Active state and function according to their definition (no failures). Several important results emerge.

There are no terminal states with one process Established and the other not established. The only terminal states have both processes Not Active (if a connection was rejected) or both processes Established. Furthermore, when both

processes are Established, sequence numbers for both directions are properly initialized as described in section II-5.2. There is no deadlock in either connection esatblishment or subsequent exchange of data.

All paths leading back to the Not Active state (except the reject path) for either process involve collisions which will cause a later retry to establish the connection. Assuming that perpetual collisions can be avoided, and that the transmission medium provides a nonzero probability of delivering any packet, the protocol will eventually succeed in establishing a connection (unless the attempt is rejected).

These results show the sufficiency of the connection establishment mechanisms embodied in the protocol machine of section A.1. Their necessity is demonstrated by theorems 4-6 in section II-5. We reprove theorem 6 in section A.5 below using the composite state formalism to show that simpler connection establishment mechanisms fail under our assumptions of worst case transmission medium behavior.

A.4 Consequences of Protocol Failures

Section A.3 has analyzed connection establishment under normal operating conditions where both protocol machines start in the Not Active state. In this section we discuss the consequences of protocol failures and the problem of

reestablishing a connection after a failure. We describe one mechanism to facilitate recovery from a common type of protocol failure. The general question of failure recovery and self-stabilizing systems [Dijkstra74] requires further research.

A common protocol failure occurs when the machine at one side of a connection "crashes," losing state information about the connection. As discussed in section II-3, it is impossible to guarantee the recovery of data in transit at the time of the failure, but it is desirable that the protocol should quickly detect the failure and reinitialize the connection for reliable communication after the failure.

After a protocol failure, we assume that all connections are placed in the Not Active state. The other side of a previously active connection may still be in the Established state. The composite state of the system will then be (NA)(ES) with some current packets possibly still being transmitted from right to left. There is no such composite state in figure A-2, indicating that this "half open" connection state can not occur in the normal protocol operation. With the current protocol specification in tables A-1 to A-4, this state also has no exit: once reached due to a failure, it is permanent. If the Established side sends data to the Not Active side, it is discarded as out-of-order (transition NA4). If the Not Active side attempts to reestablish the connection by sending a SYN packet, the Established side returns a negative acknowledgement, thinking the SYN is an old duplicate (transition ES1).

To break this stalemate, we introduce two additional protocol machine transitions:

FF1: In the Not Active state, if a data packet arrives, discard it, but return a Reject packet referencing the data packet's sequence number.

FF2: In the Established (or SYN Received) state, if a Reject packet arrives and it references a pending packet, go to the Not Active state and notify the local process that the connection is being restarted (after possible loss of pending data). Then reestablish the connection.

FIGURE A-3 ADDITIONAL COMPOSITE STATE TRANSITIONS FOR FAILURE RECOVERY

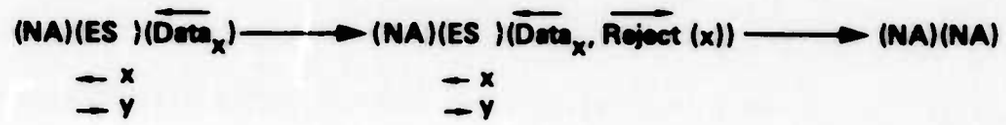


Figure A-3 shows the additions to the composite state diagram resulting from these two transitions. Without these added transitions, the Established side would eventually exceed its Quit time without knowing why. With these transitions, the protocol will restart itself (with possible loss of pending data) as long as the Established side tries to send some data after the failure. However, if the Established side is passive

(waiting to receive data), the stalemate persists. In [Cerf74b] (TCP), a Reset control packet is suggested to force return to the Not Active state in half open connections, but the later duplicate arrival of a Reset may prove dangerous.

A.5 Inadequacy of Simple Protocol

Section II-5 informally demonstrated the inadequacy of simple connection establishment mechanisms in a hostile transmission environment. Here we use composite state analysis to reprove theorem 6.

The state diagram for a simple connection establishment protocol is shown in figure A-4. Arriving SYN packets are simply accepted and acknowledged if the protocol is ready to establish a connection, (i.e. in the Not Active or SYN Sent states), or discarded as duplicates if a SYN has already been accepted (i.e. SYN Received or Established states). The protocol does not check to be sure that an arriving SYN packet is current. Collisions no longer occur since simultaneously transmitted SYN packets serve as responses to each other.

Events are a subset of those in the 3 way handshake protocol since no SYN-Verify or NACK packets exist. The Reject facility has also been removed to simplify the analysis. Table A-5 defines the complete set of transitions and actions for this simple protocol machine.

FIGURE A - 4 SIMPLE CONNECTION ESTABLISHMENT PROTOCOL MACHINE

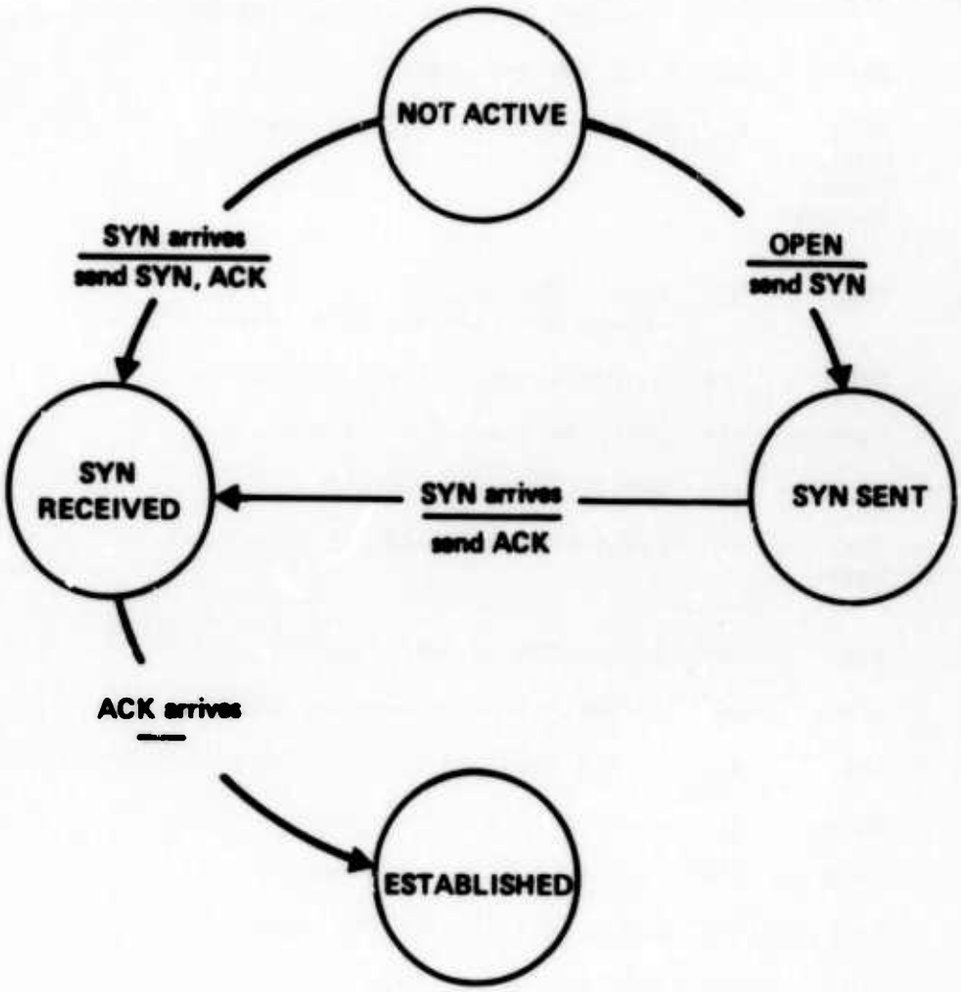


Table A-5

State Transitions for Simple Protocol Machine

Current State	Event	Next State	Action and Context
NA	SYN	SR	Accept SYN. Pick ISN and send SYN packet with ACK of received SYN.
	OPEN	SS	Pick ISN and send SYN packet.
	ACK, Quit, Data, Retrans	self	Ignore since no packets pending.
SS	SYN	SR	Accept SYN packet. Send ACK referencing received SYN.
	OPEN	self	Ignore since already in progress.
	Retrans	self	Retransmit pending SYN packet.
	Quit	self	Notify local process.
	ACK, Data	self	Discard all packets.
SR	SYN	self	Ignore SYN packet.
	ACK	ES	If ACK refers to pending SYN.
	ACK	self	If ACK does not refer to pending SYN.
	Data	self	Discard as out-of-order (or hold).
	OPEN	self	Ignore since in progress.
	Retrans	self	Retransmit pending SYN packet.
	Quit	self	Notify local process.
ES	SYN	self	Ignore SYN packet.
	ACK, Data, Quit, Retrans	self	Handle data communication as described in section II-4 (SPAR protocol).
	OPEN	self	Ignore since already in progress.

NA=Not Active, SS=SYN Sent, SR=SYN Received, ES=Established

FIGURE A - 5 COMPOSITE STATE DIAGRAM FOR SIMPLE PROTOCOL

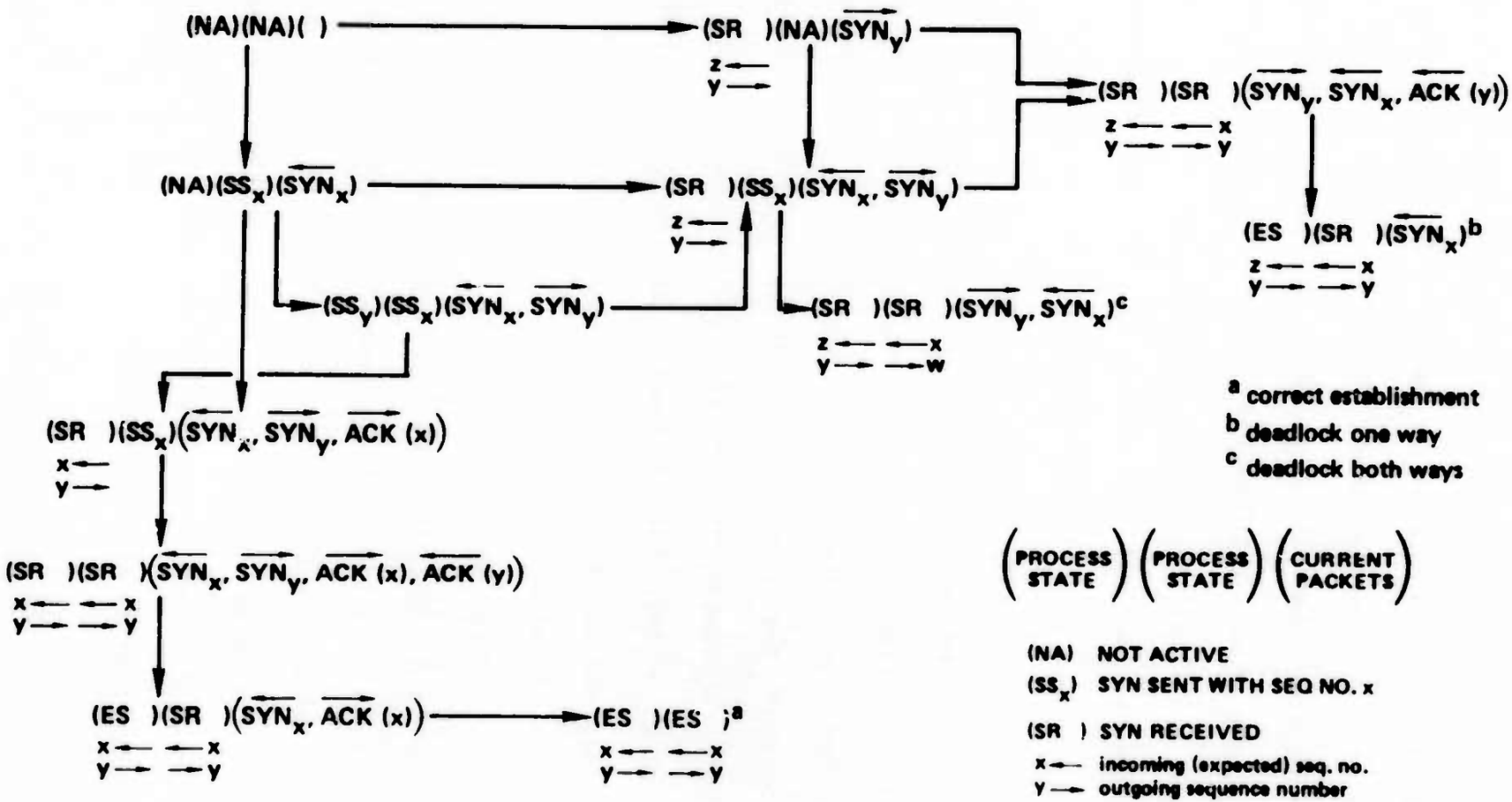


Figure A-5 shows the resulting composite state diagram using the same notation as section A.3. Assuming the protocol starts with both processes Not Active, three terminal states are possible. Correct connection establishment (state labeled "a") occurs if no old SYN packets emerge from the transmission medium while initialization is in progress. However, if one process receives an old SYN packet at an inopportune moment during connection establishment (see figure 9 in chapter II), that side of the connection will be established with incorrectly initialized sequence numbers (state labeled "b"). Old duplicate data following the old SYN may be accepted in this state. The other side of the connection will remain in the SYN Sent state because the Established side discards arriving SYN packets. If both processes receive old SYN packets during connection establishment, both processes may be trapped in the SYN Sent state and communication will be blocked in both directions (state labeled "c").

This duplicates the results obtained informally in theorem 6 that such a "credulous" connection establishment protocol is inadequate for use with hostile transmission media. Our analysis has found both an illegal state (state b allows old data to be delivered again) and a deadlock (state c is a terminal state with each process requiring a response from the other that is not forthcoming).

Appendix B

REFERENCES FOR EXAMPLE NETWORKS

ARPANET: Heart70, McQuillan72, Frank70, Ornstein72, Roberst72, Carr70, Crocker72, Crowther75, Kleinrock74a

CYCLADES: Pouzin73b, Zimmerman75

EPSS: Beeforth72, Bright74, Bright75

TYMNET: Beere71, Combs73, Tymes71

ALOHANET: Abramson70, Abramson73, Kuo73

PRNET: Burchfiel75, Kahn75, and papers from PRNET session in Proc. National Computer Conf., 1975, AFIPS Press.

UCL: Higginson75, Lloyd75b, Stokes75

DCS: Farber72, Farber72a, Farber73, Farber73a, Rowe73, Rowe75

REFERENCES

References are cited by the principal author's last name followed by the year of publication (and a letter to distinguish multiple publications from the same year).

INWG Notes are available from Technical Committee 6.1 of IFIP for the cost of reproduction: IFIP WG6.1, Vinton Cerf, Chairman, Digital Systems Lab, Stanford University, Stanford, Ca. 94305.

RFC's (Requests for Comments) and NIC documents are available from the Network Information Center in limited numbers: Elizabeth (Jake) Feinler, Stanford Research Institute, Network Information Center, 333 Ravenswood Av., Menlo Park, Ca. 94025.

[Abramson70] N. Abramson, "The ALOHA System--Another Alternative for Computer Communications," Proc. Fall Joint Computer Conf., 1970, AFIPS Press, pp. 281-285.

[Abramson73] N. Abramson, "The ALOHA System," in Computer Communication Networks, Abramson and Kuo, editors, Prentice-Hall, 1973.

[Abramson73a] N. Abramson, "Packet Switching with Satellites," Proc. National Computer Conf., 1973, AFIPS Press, pp. 695-702.

[Agnew74] C. E. Agnew, "On the Optimality of Adaptive Routing Algorithms," Proc. National Telecommunications Conf., San Diego, December 1974, pp. 1021-1025.

[Akkoyunlu74] E. Akkoyunlu, A. Bernstein, and R. Schantz, "Interprocess Communication Facilities for Network Operating Systems," Computer, June 1974, pp. 46-55.

[Akkoyunlu75] E. A. Akkoyunlu, K. Ekanadham, and R. V. Huber, "Some Constraints and Tradeoffs in the Design of Network Communications," Proc. Fifth Symp. on Operating Systems Principles, Austin, Texas, November 1975, pp. 67-74. (ACM SIGOPS Operating Systems Review 9, 5)

[Baran64] P. Baran, S. Boehm, P. Smith, "On Distributed Communication," collection of 11 papers, Rand Corporation, 1964.

[Bartlett69] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Note on Reliable Full-Duplex Transmission over Half-Duplex Links," CACM 12, 5, May 1969, pp. 260-261, 265.

[Beeforth72] T. H. Beeforth and others, "Proposed Organisation for Packet-Switched Data-Communication Network," Proc. IEE [British] 119, 12, December 1972, pp. 1677-1682.

[Beere71] M. Beere and N. Sullivan, "Tymnet--A Serendipitous Evolution," Proc. Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications, Palo Alto, California, October 1971.

[Belloni74] A. Belloni, M. Bozetti, and G. LeMoli, "Routing and Internetworking," INWG Protocol Note 10, August 1974. Also in Alta Frequenza 44, 4, April 1975, pp. 194-210.

[Belsnes74] D. Belsnes, "Flow Control in Packet Switching Networks," SU-DSL Technical Note #50 and INWG Note #63, October 1974.

[Belsnes74a] D. Belsnes, "Note on Single Message Communication," INWG Protocol Note #3, September 1974.

[Benice64a] R. J. Benice and A. H. Frey, "An Analysis of Retransmission Systems," IEEE Trans. on Communication Technology, December 1964, pp. 135-145.

[Benice64b] R. J. Benice and A. H. Frey, "Comparison of Error Control Techniques," IEEE Trans. on Communication Technology, December 1964, pp. 146-154.

[Benoit74] J. W. Benoit and D. C. Wood, "Network Front End Study," Mitre Technical Report MTR-5213, August 1974, Limited Distribution.

[Binder74] R. Binder, W. S. Lai, and M. Wilson, "The ALOHANET Menehune - Version II," ALOHA System Technical Report B74-6, September 1974.

[Binder75] R. Binder, R. Rettberg, and D. Walden, "The Atlantic Satellite Packet Broadcast and Gateway Experiments," BBN Report 3056, April 1975.

[Birke71] D. M. Birke, "State-Transition Programming Techniques and Their Use in Producing Teleprocessing Device Control Programs," Proc. Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications, Palo Alto, California, October 1971.

[Bochman74] G. V. Bochman, "Proving the Correctness of Communication Protocols," Technical Report, Département d'Informatique, Université de Montreal, November 1974.

[Bochman75] G. V. Bochman, "Logical Verification and Implementation of Protocols," Publication #190, Département d'Informatique, Université de Montreal, March 1975. Also in Proc. Fourth Data Communications Symp., Quebec City, Canada, October 1975, IEEE 75 CH1001-7 DATA, pp. 7-15 to 7-20.

[Bowdon72] E. K. Bowdon and W. J. Barr, "Cost Effective Priority Assignment in Network Computers," Proc. Fall Joint Computer Conf., 1972, AFIPS Press, pp. 755-763.

[Bredt73] T. H. Bredt, "Analysis of Operating System Interactions," Proc. AICA Congress on Theoretical Informatics, Institute for the Elaboration of Information, Univ. of Pisa, Italy, March 1973, pp. 253-281.

[Bright74] R. D. Bright, "EPSS Specification," INWG Protocol Note #1, October 1974.

[Bright75] R. D. Bright, "Experimental Packet Switch Project of the UK Post Office," in Computer Communication Networks, Grimsdale and Kuo, editors, NATO Advanced Studies Institute Series, E-4, Noordhoff Int., Leyden, Netherlands, 1975, pp. 435-444.

[Burchfiel75] J. Burchfiel, R. Tomlinson, and M. Beeler, "Packet Radio Station Hardware, Operating System, and Applications Programming Environment," Packet Radio Temporary Note 138, April 1975.

[Burton72] H. O. Burton and D. D. Sullivan, "Errors and Error Control," Proc. IEEE 60, 11, November 1972, pp. 1293-1300.

[Canada75] Five Contributions to CCITT from Canada, INWG Protocol Note #24, March 1975.

[Carr70] S. Carr, S. Crocker, and V. Cerf, "Host/Host Protocol in the ARPA Network," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 589-597.

[Cashin74] (P. Cashin) Computer Communications Group of Trans-Canada Telephone System, "Datapac: Standard Network Access Protocol," November 1974.

[Cerf73] V. G. Cerf, "Host and Process Level Protocols for Internetwork Communication," INWG Note 39, September 1973.

[Cerf74a] V. G. Cerf, and C. Sunshine, "Protocols and Gateways for the Interconnection of Packet Switching Networks," ALOHA System Technical Report CN74-22, January 1974. Also in Proceedings of the Subconference on Computer Networks, HICSSS-7, Honolulu, Hawaii, January 1974.

[Cerf74b] V. G. Cerf, Y. Dalal, and C. Sunshine, "Specification of Internet Transmission Control Program," INWG Note 72, revised December 1974.

[Cerf74c] V. G. Cerf and R. E. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Trans. on Communications, COM-22, May 1974, pp. 637-648.

[Cerf75] V. G. Cerf, A. McKenzie, R. Scantlebury, and H. Zimmerman, "Proposal for an Internetwork End to End Protocol," INWG Note #96, September 1975. (Submitted to CCITT Study Group VII)

[Cerf75a] V. G. Cerf, D. D. Cowan, R. C. Mullin, and R. G. Stanton, "Topological Design Considerations in Computer Communication Networks," in Computer Communication Networks, Grimsdale and Kuo, editors, NATO Advanced Studies Institute Series, E-4, Noordhoff Int., Leyden, Netherlands, 1975, pp. 101-112.

[Chu74] W. W. Chu, "Asynchronous Time Division Multiplexing Systems," in Computer Communication Networks, Abramson and Kuo, editors, Prentice-Hall, 1973, pp. 237-268.

[Closs73] F. Closs, "Packet Arrival and Buffer Statistics in a Packet Switching Node," IBM Research Report RZ 594 (#20102), September 1973.

[Cochi73] B. J. Cochi, "Several Stochastic Models of Computer Systems," PhD Thesis, Stanford University, August 1973.

[Coffman73] E. G. Coffman, Jr. and P. J. Denning, Operating Systems Theory, Prentice-Hall, 1973.

[Cole71] G. D. Cole, "Computer Network Measurements: Techniques and Experiments," Report UCLA-ENG-7165, Computer Science Dept., UCLA, October 1971. (PhD Thesis)

[Combs73] B. Combs, "TYMNET: A Distributed Network," Datamation, July 1973, pp.40-43.

[Cox61] D. R. Cox and W. L. Smith, Queues, Methuen, London, 1961.

[Crocker72] S. Crocker, J. Heafner, R. Metcalfe, and J. Postel, "Function Oriented Protocols for the ARPA Network," Proc. Spring Joint Computer Conf., 1972, AFIPS Press, pp. 271-280.

[Crowther74] W. Crowther and others, "Report on Network Design Issues," BBN Report No. 2918, November 1974. Also INWG Note #64.

[Crowther75] W. Crowther and others, "Issues in Packet Switching Network Design," Proc. National Computer Conf., 1975, AFIPS Press, pp. 161-175.

[Dalal74] Y. K. Dalal, "More on Selecting Sequence Numbers," Proc. ACM SIGCOMM/SIGOPS Interprocess Communications Workshop, Santa Monica, March 1975, pp. 25-36. (ACM Operating Systems Review, 9, 3, July 1975) Also INWG Protocol Note #4, October 1974.

[Dalal75] Y. K. Dalal, "Establishing a Connection," INWG Protocol Note #14, March 1975.

[Danthine75a] A. Danthine and J. Bremer, "Définition, Représentation, et Simulation de Protocoles dans un Contexte Réseaux," Journ. Intern. Mini-ordinateurs et Transm. de données, AIM, Liege, January 1975.

[Danthine75b] A. Danthine and J. Bremer, "Communication Protocols in a Network Environment," Proc. ACM SIGCOMM/SIGOPS Interprocess Communications Workshop, Santa Monica, March 1975, pp. 87-92. (Also ACM Operating Systems Review, 9, 3, July 1975)

[Danthine75c] A. Danthine and L Eschenauer, "Influence on the Node Behavior of the Node-to-Node Protocol," INWG Protocol Note #22, March 1975. Also in Proc. Fourth Data Communications Symp., Quebec City, Canada, October 1975, IEEE 75 CH1001-7 DATA, pp. 7-1 to 7-8.

[Davies72] D. W. Davies, "The Control of Congestion In Packet-Switching Networks," IEEE Trans on Communications, COM-20, 3, June 1972, pp. 546-550.

[Davies73] D. W. Davies, "Interconnection of Networks at the Packet or Host Level," Unpublished INWG Letter, September 1973.

[Davies74] D. W. Davies, Communication Networks for Computers, John Wiley, 1974.

[Day75] J. D. Day, "Resilient Protocols for Computer Networks," CAC Doc. No. 162 (Research in Network Data Management and Resource Sharing), Center for Advanced Computation, Univ. Illinois at Urbana-Champaign, May 1975, pp. 25-70.

[Day75a] J. Day, "A Note on Some Unresolved Issues for an End-to-End Protocol," INWG Note #98, August 1975.

[Dijkstra68] E. W. Dijkstra, "A Constructive Approach to the Problem of Program Correctness," BIT 8, 3, 1968, pp. 174-186.

[Dijkstra74] E. W. Dijkstra, "Self-stabilizing Systems in Spite of Distributed Control," Comm. ACM 17, November 1974, pp. 643-644.

[Donnan74] R. A. Donnan and J. R. Kersey, "Synchronous Data Link Control: A Perspective," IBM Systems J. 13, 2, 1974, pp. 140-162.

[EPSS75] EPSS Liaison Group, "A Basic File Transfer Protocol," INWG Note #93, June 1975.

[EPSS75a] EPSS Liaison Group, "An Interactive Terminal Protocol," INWG Note #94, June 1975.

[Ewin70] J. C. Ewin and P. K. Giloth, "No. 1 ESS ADF: System Organization and Objectives," Bell System Technical Journal, December 1970, pp. 2733-2752.

[Farber72] D. J. Farber and K. Larson, "The Structure of A Distributed Computer System--The Communications System," Proc. Symp. on Computer Communications Networks and Teletraffic, Microwave Research Institute of Polytechnic Institute of Brooklyn, April 1972, pp. 21-27.

[Farber72a] D. J. Farber and F. R. Heinreich, "The Structure of A Distributed Computer System--The Distributed File System," Proc. International Conference on Computer Communications, October 1972, pp 364-370.

[Farber73] D. J. Farber and J. J. Vittal, "Extendability Considerations in the Design of the Distributed Computer System (DCS)," Proc. National Telecommunications Conference, Atlanta, Georgia, November 1973.

[Farber73a] D. J. Farber and others, "The Distributed Computing System," Proc. COMPCON73, San Francisco, February-March 1973, IEEE 73CH0716-1C, pp. 31-34.

[Feinroth73] Y. Feinroth, E. Franceschini, and M. Goldstein, "Telecommunications Using a Front-End Minicomputer," Comm. ACM 16, 3, March 1973, pp. 153-160.

[Floyd67] R. W. Floyd, "Assigning Meanings to Programs," Proc. American Math Society Symp. in Applied Mathematics 19, pp. 19-31.

[Forgie75] J. W. Forgie and C. K. McElwain, "ARPANET Delay Measurements," NSC Note 70, M. I. T. Lincoln Labs, July 1975.

[Frank70] H. Frank, I. T. Frisch, and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 581-587.

[Frank71] H. Frank and W. Chou, "Routing in Computer Networks," Networks, 1, 3, 1971, John Wiley, pp. 99-112.

[Frank72] H. Frank, R. E. Kahn, and L. Kleinrock, "Computer Communication Network Design: Experience With Theory and Practice," Proc. Spring Joint Computer Conf., 1972, AFIPS Press, pp. 255-270.

[Frank72a] H. Frank and W. Chou, "Topological Optimization of Computer Networks," Proc. IEEE 60, 11, November 1972, pp. 1385-1396.

[Fultz72] G. L. Fultz, "Adaptive Routing Techniques for Message Switching Computer-Communication Networks," UCLA-ENG-7252, July 1972. (PhD THesis, UCLA)

[Gaver71] D. P. Gaver and P. A. W. Lewis, "Probability Models for Buffer Storage Allocation Problems," Jour. ACM, 18, 2, April 1971, pp. 186-198.

[Gilbert72] P. Gilbert and W. J. Chandler, "Interference Between Communicating Parallel Processes," Comm. ACM 15, 6, June 1972, pp. 427-437.

[Graham71] R. L. Graham and H. O. Pollack, "On the Addressing Problem for Loop Switching," Bell System Technical Journal 50, 8, October 1971, pp. 2495-2519.

[Heart70] F. E. Heart and others, "The Interface Message Processor for the ARPA Computer Network," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 551-567.

[Higginson75] P. L. Higginson, and A. J. Hinchley, "The Problems of Linking Several Networks with a Gateway Computer," Proc. European Computing Conf. on Communication Networks, September 1975, Online Conferences Ltd., Uxbridge, England, pp. 25-34.

[Hoare69] C. A. R. Hoare, "An Axiomatic Basis for Computer Programming," Comm. ACM 12, 10, October 1969, pp. 576-580, 583.

[Holt72] R. C. Holt, "Some Deadlock Properties of Computer Systems," Computing Surveys 4, 4, December 1972, pp. 179-196.

[INWG74] INWG Protocol Committee, "Experiment in Inter-networking. Basic Message Format," INWG Doc. 1, November 1974.

[INWG75] INWG, "Timing Parameters of Packet Switched Data Networks--User Implications," INWG Note #82, May 1975.

[INWG75a] INWG, "Basic Message Format for Internetwork Communication," INWG Note #83, May 1975.

[Kahn72] R. E. Kahn and W. R. Crowther, "Flow Control in a Resource-Sharing Computer Network," IEEE Trans. on Communications, COM-20, 3, June 1972, pp. 539-546.

[Kahn72a] R. E. Kahn, "Resource-Sharing Computer Communications Networks," Proc. IEEE 60, 11, November 1972, pp. 1397-1407.

[Kahn75] R. E. Kahn, "The Organization of Computer Resources into a Packet Radio Network," Proc. National Computer Conf., 1975, AFIPS Press, pp. 177-186.

[Kalin71] R. B. Kalin, "Achieving Reliable Communication," RFC 203, August 1971.

[Karp72] D. Karp and S. Seroussi, "A Communications Interface for Computer Networks," IEEE Trans. on Computer Communications, COM-22, 3, June 1972, pp. 550-556.

[Karp73] P. M. Karp, "Origin, Development, and Current Status of the ARPANET," Proc. COMPCON73, San Francisco, February-March 1973, IEEE 73CH0716-1C, pp. 49-52.

[Kersey74] J. R. Kersey, "Synchronous Data Link Control," Data Communications, May/June 1974, pp. 49-60.

[Kershenbaum74] A. Kershenbaum, "Tools for Planning and Designing Data Communications Networks," Proc. National Computer Conf., 1974, AFIPS Press, pp. 583-591.

[Kleinrock70] L. Kleinrock, "Analytic and Simulation Methods in Computer Network Design," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 569-579.

[Kleinrock74] L. Kleinrock, W. E. Naylor, and H. Opderbeck, "A Study of Line Overhead in the ARPANET," INWG Note #71, September 1974. To appear in Comm. ACM.

[Kleinrock74a] L. Kleinrock and W. E. Naylor, "On Measured Behavior of the ARPA Network", Proc. National Computer Conf., 1974, AFIPS Press, pp. 767-780.

[Kleinrock75] L. Kleinrock, Queueing Systems, Vol. I, John Wiley, New York, 1975.

[Kuo73] F. F. Kuo and N. Abramson, "Some Advances in Radio Communications for Computers," Proc. COMPCON73, San Francisco, February-March 1973, IEEE 73CH0716-1C, pp.57-60.

[Kuo74] F. F. Kuo, "Political and Economic Issues for Internetwork Connections," INWG Note 58, May 1974.

[Kuo75] F. F. Kuo, "Public Policy Issues Concerning ARPANET," INWG Legal/Political Note #4, June 1975.

[Lay73] W. M. Lay, D. L. Mills, and M. V. Zelkowitz, "Design of a Distributed Computer Network for Resource Sharing," Proc. AIAA Computer Network Systems Conf., Huntsville, Alabama, April 1973, paper 73-426.

[LeLann73] G. LeLann, "A General Model for Networks, Application to ARPANET and CYCLADES," unpublished notes, October 1973.

[LeMoli73] G. LeMoli, "A Theory of Colloquies," Alta Frequenza 42, 10, 1973, pp. 493-223E to 500-230E. Also presented at First European Workshop on Computer Networks, Arles, April 1973, pp. 153-173.

[LeMoli75] G. LeMoli, "A Proposal for Fragmenting Packets in Internetworking," INWG Protocol Note 21, April 1975.

[Lloyd75a] D. Lloyd, M. Galland, and P. T. Kirstein, "Aims and Objectives of Internetwork Experiments," INWG Experiment Note 3, February 1975. Also INDRA Note 3[illegible].

[Lloyd75b] D. Lloyd, and P. T. Kirstein, "Alternative Approaches to the Interconnection of Computer Networks," Proc. European Computing Conf. on Communication Networks, September 1975, Online Conferences Ltd., Uxbridge, England, pp. 25-34.

[Lynch68] W. C. Lynch, "Reliable Full-Duplex File Transmission over Half-Duplex Telephone Lines," Comm. ACM 11, 6, June 1968, pp. 407-410.

[MacPherson75] S. MacPherson, editor, "Definitions for Packet-Mode Terms Used in Proposed Recommendations," CCITT Study Group VII Paper No. 212-E, Annex 9, June 1975. Also INWG Protocol Note #27.

[Mader74] E. R. Mader, W. W. Plummer, and R. S. Tomlinson, "A Protocol Experiment," INWG Experiments Note #1, August 1974.

[McKay73] D. B. McKay and D. P. Karp, "Protocol for a Computer Network," IBM System Journal 12, 1, 1973, pp. 94-105.

[McKenzie74] A. M. McKenzie, "Internetwork Host-to-Host Protocol," INWG Note 74, December 1974.

[McKenzie74a] A. M. McKenzie, "Some Computer Network Interconnection Issues," Proc. National Computer Conference, 1974, AFIPS Press, pp. 857-859.

[McQuillan72] J. M. McQuillan, W. R. Crowther, B. P. Cosell, D. C. Walden, and F. E. Heart, "Improvements in the Design and Performance of the ARPA Network," Proc. Fall Joint Computer Conf., 1972, AFIPS Press, pp. 741-754.

[McQuillan74] J. M. McQuillan, "Adaptive Routing Algorithms for Distributed Computer Networks," BBN Report No. 2831, May 1974. (PhD Thesis, Harvard University)

[Merlin74] P. M. Merlin, "A Study of the Recoverability of Computing Systems," Technical Report #58 (PhD Thesis), University of California at Irvine, November 1974.

[Merlin75] P. M. Merlin and D. J. Farber, "Recoverability of Communication Protocols--Implications of a Theoretical Study," IBM Research Report RC 5416 (#23664), Yorktown Heights, N.Y., May 1975.

[Metcalfe72] R. M. Metcalfe, "Strategies for Interprocess Communication in a Distributed Computing System," Proc. Symp. on Computer Communications Networks and Teletraffic, Microwave Research Institute of Polytechnic Institute of Brooklyn, April, 1972, pp. 519-526.

[Metcalfe73] R. M. Metcalfe, "Packet Communication," M.I.T. Project MAC Report TR-114, December 1973. (PhD Thesis, Harvard University)

[Naylor73] W. E. Naylor, Real Time Transmission in a Packet Switching Network," Network Measurment Center Note #15, September 1973. Also NIC 19014

[Naylor75] W. E. Naylor, "A Loop-free Adaptive Routing Algorithm for Packet Switched Networks," Proc. Fourth Data Communications Symp., Quebec City, Canada, October 1975, IEEE 75 CH1001-7 DATA, pp. 7-9 to 7-14.

[Naur66] P. Naur, "Proof of Algorithms by General Snapshots," BIT 6, 4, 1966, pp. 310-316.

[Newport72] C. B. Newport and J. Ryzlak, "Communication Processors," Proc. IEEE 60, 11, November 1972, pp. 1321-1332.

[Opderbeck74] H. Opderbeck and L. Kleinrock, "The Influence of Control Procedures on the Performance of Packet-Switched Networks," Proc. National Telecommunications Conf., San Diego, December 1974. Also INWG Note #62, September 1974.

[Ornstein72] S. M. Ornstein and others, "The Terminal IMP for the ARPA Computer Network," Proc. Spring Joint Computer Conf., 1972, AFIPS Press, pp. 243-254.

[Padlipsky74] M. Padlipsky, "A Proposed Protocol for Connecting Host Computers to ARPA-Like Networks via Directly Connected Front End Processors," RFC #647, NIC 31117, November 1974.

[Pierce72] J. R. Pierce, "Network for Block Switching of Data," Bell System Technical Journal 51, 6, July-August 1972, pp. 1133-1145.

[Postel74] J. B. Postel, "A Graph Model Analysis of Computer Communications Protocols," UCLA-ENG-7410, January 1974. (PhD Thesis, UCLA)

[Pouzin73] L. Pouzin, "Interconnection of Packet Switching Networks," INWG Note 42, October 1973. Also SCH 513.1

[Pouzin73a] L. Pouzin, "Efficiency of Full-Duplex Synchronous Data Link Procedures," INWG Note #35, June 1973. Also NIC 18255.

[Pouzin73b] L. Pouzin, "Presentation and Major Design Aspects of the CYCLADES Computer Network," Proc. Third Data Communications Symp., Tampa, November 1973, pp. 80-85. Also NIC 18256, INWG Note #36, and SCH 511.

[Pouzin74a] L.Pouzin, "Interconnection of Packet Switching Networks," ALOHA System Technical Report CN74-21, January 1974. Also in Proc. Subconference on Computer Networks, HICSS-7, Honolulu, Hawaii, January 1974.

[Pouzin74b] L. Pouzin, "A Proposal for Interconnecting Packet Switching Networks," INWG Note 60, March 1974. Also SCH 527.

[Pouzin74c] L. Pouzin, "Basic Elements of a network data link control procedure (NDLC)," INWG Note 54, NIC 30375, January 1974.

[Pouzin75] L. Pouzin, "Virtual Call Issues in Network Architectures," INWG Note #92, June 1975. (To be presented at EUROCOMP, Brunel Univ., September 1975.)

[Pyke73] T. N. Pyke and R. P. Blanc, "Computer Networking Technology--A State of the Art Review," Computer, August 1973, pp. 13-19.

[Roberts70] L. G. Roberst and B. D. Wessler, "Computor Network Development to Achieve Resource Sharing," Proc. Spring Joint Computer Conf., 1970, AFIPS Press, pp. 543-549.

[Roberts72] L. G. Roberts and B. Wessler, "The ARPA Computer Network," in Computer Communication Networks, Abramson and Kuo, editors, Prentice Hall, 1972.

[Rowe73] L. A. Rowe, M. D. Hopwood, and D. J. Farber, "Software Methods for Achieving Fail-Soft Behavior in the Distributed Computing System," Proc. IEEE Symp. on Computer Software Reliability, New York, April/May 1973, pp. 7-11.

[Rowe75] L. A. Rowe, "The Distributed Computing System," Tech. Report #66, Dept. Information and Computer Science, Univ. California, Irvine, June 1975.

[Sastry74] A. R. K. Sastry, "ARQ Error Control on Satellite Channels with High Error Rates," Technical Report TR-327, University of Maryland Department of Computer Science, September 1974.

[Seidler75] J. Seidler, "A Method of Analysis of Discrete Communication Feedback Systems," Information and Control 29, 2, October 1975, pp. 125-140.

[Stokes75] A. V. Stokes, and P. L. Higginson, "The Problems of Connecting Hosts into ARPANET," Proc. European Computing Conf. on Communication Networks, September 1975, Online Conferences Ltd., Uxbridge, England, pp. 25-34.

[Stutzman72] B. W. Stutzman, "Data Communication Control Procedures," Computing Surveys 4, 4, December 1972, pp. 197-220.

[Sunshine74] C. A. Sunshine, "Issues In Communication Protocol Design--Formal Correctness," INWG Protocol Note #5, October 1974.

[Thomas73] R. H. Thomas, "A Resource Sharing Executive for the ARPANET," Proc. National Computer Conference, 1973, AFIPS Press, pp. 155-163.

[Tomlinson74] R. S. Tomlinson, "Selecting Sequence Numbers," Proc. ACM SIGCOMM/SIGOPS Interprocess Communications Workshop, Santa Monica, March 1975, pp. 11-23. (ACM Operating Systems Review, 9, 3, July 1975) Also INWG Protocol Note #2, August 1974.

[Townsend64] R. L. Townsend and R. N. Watts, "Effectiveness of Error Control In Data Communications over the Switched Telephone Network," Bell System Tech. J. 43, 6, November 1964.

[Trafton71] P. J. Trafton, H. A. Blank, and N. F. McAllister, "Data Transmission Network Computer-to-Computer Study," Proc. Second ACM/IEEE Symp. on Problems In the Optimization of Data Communications, Palo Alto, Ca., October 1971, pp. 183-191.

[Tymes71] L. R. Tymes and T. W. Hall, "TYMNET--A Terminal Oriented Communication Network," Proc. Spring Joint Computer Conf., 1971, AFIPS Press, pp. 211-216.

[UKPO74] United Kingdom Post Office, "Some Considerations on Flow Control for International Packet Transport," INWG Protocol Note #8, June 1974.

[VanSlyke72] R. VanSlyke and H. Frank, Network Reliability Analysis--I," Networks 1, 3, 1972, John Wiley pp. 279-290.

[Walden72] D. C. Walden, "A System For Interprocess Communication in A Resource Sharing Computer Network," Comm. ACM, 15, 4, April 1972, pp. 221-230.

[Walden74] D. C. Walden, "Some Changes to the IMP and the IMP/HOST Interface," RFC 660, October 1974.

[Watson73] R. W. Watson, "Some Thoughts on System Design to Facilitate Resource Sharing," RFC #592, November 1973.

[Weber64] J. H. Weber, "A Simulation Study of Routing and Control in Communication Networks," Bell System Tech. J. 43, 6, November 1964.

[Zimmerman73] H. Zimmerman and M. Elie, "Transport Protocol. Standard Host-Host Protocol for Heterogeneous Computer Networks," INWG Note #61, June 1973. Also SCH 519.1.

[Zimmerman75] H. Zimmerman, "The CYCLADES End-End Protocol," Proc. Fourth Data Communications Symp., Quebec City, Canada, October 1975, IEEE 75 CH1001-7 DATA, pp. 7-21 to 7-26.